PANEL'92

XVIII CONFERENCIA LATINOAMERICANA DE INFORMATICA

ACTAS

Universidad de Las Palmas de Gran Canaria

CLEI
Centro Latinoamericano de Estudios en Informática

CEDINFOR

A.O
17
92

CLEI
Centro
Latinoamericano
de Estudios en
Informática

Universidad de Las Palmas
de Gran Canaria

PANEL'92

XVIII CONFERENCIA LATINOAMERICANA DE INFORMATICA

ACTAS

FUNDACION CEDINFOR
ORIGEN
DOCUMENTO
NRO. INV 1280
FECHA
16 DIC 1993

Las Palmas de Gran Canaria, ESPAÑA

31 de Agosto al 4 de Septiembre de 1992

Tabla de Contenidos / Table of Contents

MacEmulus: Entorno Amistoso para el Acceso a Servicios Telemáticos
Abal, F. 1

Aviliação dos Métodos de Aquição de Conhecimento no Projeto GEOXPERT
Abel, M.; de Rocha C., A. C.; Della F., J. C. 9

Um Ambiente de Suporte ao Trabalho Cooperativo
Aguiar, C. A. T.; Salgado, A. C. 17

A Shema Modification Methodology for an Object-Oriented Database System
Al-Hajj, R.; Arkun, M. E. 26

Edición Gráfica para Invidentes
Alonso A., F.; Montes, C.; Pazos, J. 34

Compilación Optimizante para el Lenguaje Occam sobre un Sistema Time-Sharing
Alvarez, G. A.; Bandinelli, D. O.; Fernández, M. O.; Rosini, G. A. 44

Un Nuevo Paradigma de Enseñanza para la Disciplina de Computación
Alvarez R., J. 52

Técnicas de Procesamiento Paralelo para Sistemas Gráficos de Alto Desempenho
Anido, M. L. 59

A CAD System for Teaching the Design of VLSI Circuits - Status and Evolution -
Anido, M. L.; Oliveira, C. E. T. 70

Etiquetado de Artículos Científicos: un primer intento de construcción de un sistema basado en el conocimiento para tratar ficheros ASCII para publicar con Ventura
Aramburu C., M. J.; Hammersley, P. 79

A family of cuts for the matroidal knapsack problem
Araoz, J.; Maffioli, F. 90

Un Modelo de Hoja de Cálculo: Observaciones a Implementaciones
Arraiz, E.; Soler, R.; Zoltan, C. 98

Gilena: Generador de Interfaces en Lenguaje Natural
Atkinson A., J. 106

Un Modelo de Evaluación del Desempeña de Sistemas Cliente-Servidor
Aurtenechea, F.; Rosenberg, G; Casas, I. 118

CIG: A Class Interface Generator for C++
Baeza-Yates, R.; Tichy, M. 129

Generalización del Operador de Cruce en Algoritmos Geneticos
Barrios, D.; Rios, J.; Segovia, J. 137

Métodos para el cálculo de la Función Descriptiva en sistemas dinámicos no lineales
Bellassai, G.; Ferrari, L. 145

Stack Structure for the efficient execution of the FORTH Languaje Instruction Set
Bernal N., A. 153

Extensión del Modelo de Sistemas Expertos usando Aprendizaje Automático
Bocic, S. A.; Fuller, D. A. 162

Prototyping Action Semantics Using Functional Languajes
Borba , P.; Meira, S.; Santos, A. 176

Suporte de Bancos de Dados para Sistemas de Hipertextos
Borges, M. R. S. 184

Specifications from Multiple Viewpoints: The Temoral-Causal Way
Brelaz de Castro, J. F. 193

Adquisición de Nuevo Conocimiento por Comparación y Construcción de la Jerarquía de Clases Formales
Briand, H. 204

ÑERANDU: Lenguaje de Soporte de Decisiones en la Industria
Cabello, C.; Aranda, S.; Rivaldi, M. 213

Writing Tools for Non-Native Users of English
Caldeira, S. M. A.; De Oliveira, M. C. F.; Fontana, N. M.; Nacamatsu, C. Y.; Oliveira Jr.; O. N. 224

An Overiew of Software Rosk Management Methods
Calvo-Manzano, J. A.; Maté, J. L.; San Feliu, T. 232

Un Sistema de Visión para la Detección de Defectos
Candela, S.; Muñoz, J. A.; Alayón, F.; García, C. 240

DIARIA: An Integrated Inference Engine
Caraça-Valente, J. P.; Gómez-Pérez, A.; Jurısto, N.; Morant, J. L. 248

Especificaicón y modelaje de diseños en CAD
Cardoso, R.; Cardoso, O. L.; Hernández, J. T.; Ramos, O. 255

Weightless Neurona Models
Carvalho F., E. C. B. 265

Uma Linguagem Pseudo-Natural para Construção de Interfaces Multimídia
Casado, M. G.; Salgado, A. C. 274

Design and Cooperation Management Through Design Steps and Object Characteristics
Castro L., M. A.; Iochpe, C. 284

Dynamic Multimodal Function Optimization using Genetic Algorithms
Cedeño, W. 292

Una Metodología Orientada por Objetos para el Diseño de Aplicaciones Gráficas Interactivas
Chicaiza S., L. B.; Villalobos S., J. A. 302

Un Algoritmo Distribuido de Detección del Interbloqueo para el Modelo Unico-Recurso
Córdoba, A.; González de Mendívil, J. R.; Garitagoitia, J. R. 311

Design of a Pipelined Multiplier Chip as the Kernel of a Vector Processing Architecture
Couto B., D. A. 319

Resolución de Ecuaciones en Derivadas Parciales Mediante Algoritmos Paralelos
Crespo F., J. L.; Mora M., E.; Peire A., J. 328

CSP Probabilístico (PCSP): Un modelo operacional
Cuartero G., F.; de Frutos E., D.; Valero R., V. 338

Derivación y Verificación de Algoritmos en Control Numérico a partir de una Interfaz Gráfica
De Giusti, A.; Lanzarini, L.; Naiouf, M.; D'Argenio, P. 347

...

Integrating Graphic Software - The Making of "M.A.T.E."
Delgado O., S.; Casacurta, A.; Schimidt, A. E.; Laschuk, A.; Dal Sasso F., C. M.; de Oliveira, J. B.; de Oliveira N., M. M.; Kordyas D., M.; Nascimento, M. E.; Rodriguez L., R.; Bastos, R.; Raupp M., S. 353

Geohpar - Resena Geográfica e Histórica del Paragay
Díaz L., H. 362

A Goal Model for the Evaluation of Videotex Systems
Domínguez, E.; Escario, I.; Zapata, M. A.; Lapeña, M. J.; Miguel, A. ... 368

Digitized Segments and Straight Lines
Domínguez, E.; Frances, A.; Márquez, A. 376

Druida: Una Herramiento Basada en Conocimiento para el Afinamiento de Sistemas Unix
Droguett C., M.; Casas, I.; Aurtenechea O., F. 384

Reconocimiento de Palabras Aisladas utilizando Modelos Ocultos de Markov
Escrig M., M. T.; Casacuberta N., F. 395

Adaptive Parallel Sorting
Estivill-Castro, V. 403

TERAM ALGORITHMS Uma Aplicação para Fluxo de Potência Eléctrica
Falcão, D. M.; Barán, B. 413

Communicating Application Management: A Service-Oriented Server Modelling
Faure, F.; Desprats, T.; Raynaud, Y. 421

Cache Memory for a LISP Machine
Feitosa, R. Q.; Guidacci da Silveira, G. F. 433

Revisión Mutua de Creencias: Aplicación a Bases de Conocimiento
Fermé, E. L.; González, O. O. 443

Umbralización Adaptativa en Tiempo Real
Fernández, M. A.; Mira, J. 452

The Binary-Radix Bucket-Region-Directory: A Simple New Directory for the Grid File
Finke, U.; Hinrichs, K.; Becker, L. 460

Problemática de la Adquisición Inductiva de Conocimiento a Partir de Ejemplos
Fiol, G.; Miró-Nicolau, J. 473

Aplicación de CLP para Mantener la Razón en un Sistema de Control con Restricciones
Forradellas, R.; Berlanga, R.; Ibañez, F.; Barber, F.; Toledo, F.; Martín, G. 483

Hacia la generación de reconocedores LR(1) de reducidas dimensiones
Fortes G., J. 493

PERSEO: Ambiente de Programación de Aplicaciones Transaccionales Distribuidas sobre Bases de Datos Relacionales
Franky, C.; Abasolo, J.; Cucalon, R.; Acosta, G.; Maya, S. 501

Towards Efficient Partial Evaluation in Logic Programming
Fuller, D. A.; Bocic, S. A.; Bertosi, L. E. 510

Evaluating the cost of the dnamic reconfiguration of a multicomputer network
García, J. M.; Duato, J. 523

The Proto-Architecture of WOLF, a Dataflow Supercomputer
García Neto, A.; Ruggiero, C. A. 531

Otimizações Importantes para Paralelismo OU em Prolog sobre Máquinas com Memória Distribuída
Geyer, C.; Briat, J.; Favre, M.; Chassin de K., J. 540

Aplicação de Paralelismo na Sístese de Imagens Através do Método da Radiosidade
Guimarães B.; G. V.; de Carvalho G., L. 548

Selecting Vectors from a Training Set for Backpropagation Networks
Gutierrez, A. 555

A Cost Evaluator for Parallel Deductive Detabase Systems
Hameurlain, A. 563

Using Communications and Multimedia Technologies in a New Approach for Computers in Education in Developing Countries
Hepp K., P.; Alvarez H., M. I.; Hinostroza S., E.; Laval M., E. 567

Criterios de Cobertura para Pruebas de Programas Paralelos
Hernández, E.; Teruel, A. 575

Paralelismo en Visualización Volumétrica
Hung, A.; Teruel, A. 583

Problema de Scheduling resuelto mediante un Sistema de Restricciones Temporales
Ibañez, F.; Berlanga, R.; Forradellas, R.; Barbe, F.; Toledo, F.; Martín, G. 592

A Fast Systolic Algorithm for Matrix Multiplication
Isern, G. 603

IABD: Diseño e Implementación de un API para el Acceso a un Sistema Manejador de Bases de Datos Relacional desde un programa C
Jiménez G., C. L.; Blanco M., R. 614

BIER: The Executable Conceptual Data Model Based on an Object Oriented Database System
Kastner, F. 624

Algoritmos Paralelos Adaptativos de Ordenamieno
Koufaty, D. A. 628

Animação da Dinâmica do Movimiento e Colisões de Corpos Rígidos
Lemos, R. R.; de Bastos, R. M. R.; Freitas, C. M. D. S.; Wagner, F. R. ... 636

DisCo: Um Ambiente de Programação Distribuída com Configuração Dinâmica de Processos
Lima Filho, J. A.; Freire C., P. R. 644

Programming the Asynchronous Polycyclic Architecture
Lino de Campos, G.; Getschko, D. 653

The Use of a Semi-Formal Specification Language to Bridge the Gap between Informal and Formal Specifications of Spftware Systems: A Proposal
Loh, S.; Castilho, J. M. 661

Representação de Conhecimentos em um Modelo de Dados Orientado a Objetos
Lopes M., O. M.; Min, L. T.; Schechtman S., S.; Antunes S., J. S. 669

Ada Reusable Components for the Construction of Robust Programs by Means of Algebraic Specifications of Abstract Data Trypes
Losavio, F. 678

An Alternative Logic Programming Execution Model for Graphical Debugging
Loyola, D. 689

Um Algoritmo para a Unificação no Lógica de Ordem Superior
Ludwing, A.; do Amaral, W. C. 696

Una Alternativa Eficiente para Evaluación de Restricciones de Integridad en Bases de Datos Lógicas
Márquez, A.; Tineo, L. 706

Towards The Automatic Validation of Conceptual Specifications for Database Systems - A Formal Approach
Martini, A. R.; de Caastilho, J. M. V. 714

MACOBJ - A Class-Based Tool For Numerical Problem Solving
Martini, A. R.; Pinho, M. S.; Diverio, T. A. 722

Dynamic Properties of Statecharts: A Reachability Tree and Analysis of Some Properties
Masiero, P. C.; Boaventura, I. G.; Maldonado, J. C. 728

Un Nuevo Método de Representación de Objetos en Imágenes Multinivel mediante un Grafo Dirigido Etiquetado
Medina, R.; Calvo, A.; Romero, J. A.; Fernández, N. 736

EL SETDC: un sistema basado en el modelo TDC
Medina P., J. E. 744

SIRNEM: A Parallel Neural Network Simulation System
Meira Jr., W. 757

LDBA: Lenguaje Distribuido Basado en Actores
Mendieta, F. J.; Quintero, A. 762

Categorical Models for Relevant Logics
Meré, M. C.; de Paiva, V. V. C. 771

A Dinamic Tester for use with osi Networks
Modiri, N. 779

A Protocol Analyser for the Monitoring and Analysis of osi Networks
Modiri, N. 792

Algoritmos de Comunicación en Hipercubos de Base B
Mora, C. B.; Teruel, A. 812

Un Marco Formal para la Programación Implícita de Robots
Morales V., J. E.; Villalobos S., J. A. 820

SOFTWARE as an export product from developing countries: Are there any real chances?
Moreira da C., E. 829

Implementation of an Artificial Neural Network for the Segmentation of Texture Images
Mosquera, A.; Cabello, D.; Carreira, M. J.; Penedo, M. G. 838

Sistemas de Tableau com Unificação para Lógicas Modais
Mota do Carmo C., M. 846

Um Ambiente de Resolução de Problemas para Sistemas Baseados em Conhecimiento
Mueller, A. 854

coSARA: A Computer-based Environment for Collaborative Desin, Analysis and Evaluation of Computer Systems
Mujica, S.; Eterovic, Y.; Berson, S.; Estrin, G.; Leung, P.; Tou, I.; Wu, E. 869

The Sun RPC Languajes Semantics
Musicante, M. 885

CANGA - Um Sistemas de Aquisição de Dados para Análise do Comportamento Animal em Pastejo
Musse, J. I.; Fortes R., L.; Prates, E. 893

ANDES: Un Analizador de Desempeño para Programas Paralelos
Naím, O.; Teruel, A. 901

CC++: CC++ Concurrente
Nava, A.; Chiariotti, E. 909

Métodos de Aproximación para Problemas de Planificación
Nussbaum, M.; Laval, E.; Supúlveda, M. 917

Un algoritmo paralelo de descomposición LU para matrices 'sparse'
Ortega F., M.; Reinoza, A. 925

Trabalho cooperativo e correio electrônico: uma experiência real no projeto ProTem-CNPq
Palazzo M. de O., J.; Hoppen, N.; Ramalho de A., E. E. 933

Evaluación del Desempeño de sistemas Unix en ambiente distribuido
Pavez F., E.; Urrea R., G. 943

Reasoning with Exceptions First
Pequeno, M. C. 953

Presentation of AMARGO: An Hierarchical VLSI Physical Layout Systhesis System
Perira, C. E.; Couto B., D. A. 961

Explanation Constraints in a Multi-Agent Environment
Piquemal-Baluard, C.; Glize, P. 969

O Desenvolvimiento de um Sistema de Diretóri
Pirmez, L. 980

Modelagem e Animação de Superfícies Deformáveis
Porcher N., L. 989

MHC - A Connectionist architecture for hybrid training
Ramírez, J. M. 997

Derivaciones Perezosas en Programación Lógica con Restricciones Ecueacionales
Ramis, M. J.; Piris, J.; Alpuente, M. 1008

Herramienta Gráfica Interactiva para Métodos de Elementos Finitos en 3 Dimensiones
Rivara, M. C. 1016

Empaquetamiento de Patrones. Algoritmos de Posicionamiento en Frontera
Rizo A., R.; Crespo L., A. 1025

Performance of a Phonetic Encoding Scheme for Speech Recognition Using Neural Netcorks
Rodellar, V.; Gómez, P.; García, C.; Naharro, F.; Pérez, M.; Gonzalo, C. 1041

Un Algoritmo de Marcha para la Intersección de dos Superficies
Rodríguez, O. 1049

Integración y Complejidad en Sistemas Multiexpertos
Sanfeliu, A. 1058

Revista en Hipermedia. Modelos y Experiencias
Schiavoni, A.; Queiruga, C.; Hourqueby, M.; Dómini, S.; Das Neves, F. 1080

A Dense Instruction Set Computer Architecture Proposal
Schoepke, O. S. *1088*

Aplicaciones Matemáticas en el Diseño de Sistemas Teleinformáticos
Silva R., T.; Pineda Q., M. *1096*

QUICK-DB: Um Ambiente para Projeto e Manipulação de Bases de Dados
Silveira, P. M. *1111*

Rumo à Portabilidade de Componentes de Diálogo
Soares P., M. *1119*

Un Modelo Paralelo para o Aprendizado em Redes Neuronais Baseado em Algoritmos Geneticos
Solar, M. *1126*

Um Sistema Especialista para auxiliar a Gerência de Falhas em Redes de Computadores
Specialski, E.; Komosinski, L.; Thirty, M. *1141*

Develop Distributed Applications Under the Object Oriented Model
Rakahashi, T. *1149*

An M.Sc. Level Course of Studies in Parallelism
Teruel, A.; Lentini, M.; Figueira, C. *1158*

Desparity-Based Photometric Stereo
Torreão, J. R. A.; Pimentel, C. J. L. *1166*

Tecnología de Inteligencia Artificial en las Arquitecturas para Sistemas de Control
Torres H., I. *1182*

Integración de Modelos de Desempeño en una Metodología de Desarrollo de Sistemas
Valderruten, A.; Raynaud, Y. *1192*

Fold/Unfold Transformation of Knowledge Bases under Approximate Reasoning
Valiente, G. *1202*

GAS - Gerador de Analisadores Sintácticos
Varela F., O. J.; de Lucca, J. E.; Linhares de O., E. *1210*

Un Entorno Integrado de Programación para el Lenguaje Funcional Hope$^+$
Velázquez I., J. A.; Belmonte A., M.; Castillo S., J. A. *1218*

Modelo Entidad Relación Extendido + DF
Vidal, M. E.; Márquez, A. *1227*

Local Nets
Yankelevich, D. *1235*

Compiling Timed Algebras into Timed Automata
Yovine, S. *1243*

Viewpoint Oriented Real-Time Parallel Systems Development
Zhang, L.; Galindo, M.; Marquie, D.; Raynaud, Y. *1251*

VSKEP-EDITor: A Visual Tool for Editing Reusable Skeletal Plans
Zhaohui, W.; Ansgar, B. *1259*

MacEmulus: Entorno Amistoso para el Acceso a Servicios Telemáticos

Fco. Abal, Josu Aramberri, Javier Lasa, Julen Lasa, José Miguel*

Departamento de Arquitectura y Tecnología de Computadores
Facultad de Informática
Universidad del País Vasco - Euskal Herriko Unibertsitatea
Apdo. 649, 20080 San Sebastián (España)

Resumen

La red SPRITEL es una red multinorma de valor añadido que facilita el acceso a servicios telemáticos. Estos servicios requieren la utilización de un terminal de texto (VT-100) o gráfico (CEPT1, CEPT2).

MacEmulus es un conjunto de programas que facilitan el acceso a SPRITEL, ofreciendo emulaciones de terminal para las tres normas mencionadas. Además ofrece la posibilidad de superponer un entorno con el interface propio Macintosh a aplicaciones de correo electrónico y teleconferencias. Su utilización presenta como ventajas la necesidad de un único terminal (Macintosh) frente a tres, y el uso de interfaces de usuario amistosos en vez de los interfaces de comandos de algunos servicios.

Palabras clave: Comunicación de datos, Redes de computadores, Emulación de terminal, Correo electrónico, Interfaz persona-computador

1. La red Spritel

La red Spritel es una red multinorma (ASCII VT-100, Videotex CEPT1, Videotex CEPT2) desarrollada por la Sociedad para la Promoción y Reconversión Industrial (SPRI) para proporcionar servicios telemáticos en el País Vasco [1].

Dentro del marco de Spritel se facilita el acceso a una serie de servicios, algunos soportados por la propia SPRI (correo electrónico, conferencias) y otros ofrecidos por una serie de compañías independientes que usan Spritel como sistema de acceso; entre ellos se encuentran: información de las Administraciones Públicas, información de prensa, documentación jurídica, correo electrónico europeo sobre I+D, bases de datos comunitarias, diccionarios técnicos, etc. Cada uno de esos servicios requiere un tipo específico de terminal: mientras que para algunos es suficiente un terminal orientado a caracteres, otros precisan de un terminal tipo videotex, preferentemente con color.

---

* Este trabajo ha sido financiado por el programa Spritel de la Sociedad para la Promoción y Reconversión Industrial, S.A. y por la Diputación Foral de Gipuzkoa.

La conexión con Spritel es posible de dos formas diferentes:

- Mediante red telefónica conmutada. En cada capital de la Comunidad Autónoma del País Vasco existe un punto de acceso al cual se puede conectar un usuario usando un modem. En San Sebastián, dicho punto de acceso se encuentra en la Facultad de Informática de la UPV/EHU.

- A través de la red de datos pública Iberpac-X.25.

De entre estas dos formas de acceso, la más habitual es la red telefónica conmutada, ya que es poco común que un usuario de un computador personal disponga de un enlace directo a X.25.

2. Euskom

Una de las aplicaciones básicas de Spritel, y la primera que entró en servicio, es el sistema de correo electrónico y teleconferencias Euskom, basado en el software PortaCOM[1] para un VAX 8250 [2, 3]. El objetivo de Euskom es ofrecer un medio de comunicación ágil y efectivo tanto para personas como para grupos, que pueda incluso servir de base para intercambio electrónico de documentos. Para acceder a Euskom es necesario un terminal compatible VT-100.

Euskom posibilita el envío de mensajes de persona a persona y la comunicación de grupos. El sistema de correo electrónico es el habitual: cada usuario dado de alta dispone de un nombre Euskom y de un buzón; puede enviar mensajes a los buzones de otros usuarios y a su vez Euskom almacena en su buzón los mensajes recibidos para él. En cuanto al envío de mensajes, el destinatario será, habitualmente, otro usuario Euskom (esto es, su buzón). Pero también resulta posible enviar mensajes a cualquier usuario de un fax o de la red télex. El propio sistema de correo se encarga de hacer las transformaciones necesarias a los mensajes y de realizar el proceso de establecimiento de llamada y envío, con todos los reintentos que sean necesarios.

Para comunicación entre grupos existen una serie de *conferencias* Euskom. Hay conferencias públicas, a las que puede suscribirse cualquier usuario, y conferencias privadas a las que un usuario puede suscribirse mediante petición a Spritel. Un usuario puede enviar notas a una conferencia y recibir todas las notas que otros usuarios hayan enviado (filtrado previamente por el moderador de la conferencia, si lo hay).

Una de las mejores características de Euskom es la flexibilidad que aporta a la hora de revisar el correo recibido o las notas de las conferencias a las que se está suscrito, pudiéndose realizar búsquedas de mensajes por múltiples criterios: remitente, tema, contenido, fecha, etc. Sin embargo, dado el interfaz de usuario disponible, no es tarea trivial confeccionar una sentencia de revisión.

---

[1] © 1987 KOMunity software AB

3. Otros servicios disponibles. Normas

Como ya se ha mencionado en §1, Euskom no es el único servicio al que se puede acceder a través de Spritel. A continuación enumeramos algunos de los más significativos:

- BOLSAGVC. Información bursátil en tiempo real. Gratuito. Terminal Ibertex
- COMPUSERVE. Servicio de información personal. Pago. Terminal VT-100
- CSIC. Información científica y multidisciplinar. Pago. Terminal VT-100
- DELFOS. Ayudas directas a la empresa. Gratuíto. Terminal VT-100
- EUROKOM. Correo electrónico sobre I+D. Pago. Terminal VT-100
- GPAZ. Páginas azules electrónicas. Pago. Terminal Ibertex
- INFOSERVICE. Informes comerciales. Pago. Terminal Ibertex
- MINITELNET. Acceso a la red telemática francesa (cerca de 10.000 servicios). Pago. Terminal Minitel
- PAE. Páginas amarillas electrónicas. Gratuíto. Terminal Ibertex

Y un largo etcétera.

4. MacEmulus

La existencia de sistemas de visualización según diferentes normas supone la posibilidad de ofrecer un amplio abanico de servicios, pero exige que un usuario disponga de un conjunto diverso de terminales, lo cual no es habitual. Si es habitual, sin embargo, que un usuario (empresa, investigador, usuario particular) disponga en su puesto de trabajo o en su domicilio de un computador personal equipado con un modem.

Dado el gran parque existente de computadores personales, Spritel se propuso desde el primer momento ofrecer a sus usuarios programas que, bajo plataformas PC y Macintosh, tuviesen la posibilidad de conectarse a la red vía módem, emulando además los terminales de texto o gráficos que necesitan los servicios.

MacEmulus es fruto de esa idea, y consiste en tres programas con un aspecto similar que, aparte de posibilitar la conexión y emular terminales:

- ASCII VT-100
- Videotex CEPT1 (norma Ibertex)
- Videotex CEPT2 (norma Minitel)

permiten la utilización de las funciones de los programas a través de los elementos que se han hecho habituales en un GUI [4] (*Graphical User Interface*): ratón como dispositivo apuntador, menús de persiana, presentación multiventana, iconos, etc.

La disponibilidad en un computador de dispositivos de almacenamiento de información han posibilitado dotar de cierta "inteligencia" a los terminales emulados, habiéndose añadido posibilidades como:

- Captura de sesiones
- Posibilidad de "volver atrás" y revisar una sesión
- Captura de secciones de texto y/o gráficos ("cut") con la posibilidad de utilizarlas ("paste") en otras aplicaciones Macintosh
- Operación desatendida mediante un lenguaje de comandos

Una aplicación básica en Spritel es Euskom, y a ella se ha prestado especial atención. Dadas las dificultades que presenta su interface de usuario, se ha superpuesto sobre él un GUI con el estilo Macintosh que facilita el uso de todas sus opciones mediante selecciones en un conjunto de menús.

Como sistema de desarrollo se han utilizado computadores Macintosh Plus, II, IIx y IIfx, con los lenguajes de programación V.I.P. [6] y C [7]. V.I.P. es un lenguaje interpretado que posibilita un acceso razonablemente sencillo al sistema operativo del Macintosh [5], pero que no resulta viable para preparar programas aptos para ser distribuídos (lentitud, disponibilidad del intérprete). Sin embargo, un programa V.I.P. puede ser traducido automáticamente a un equivalente en C, lo que permite obtener programas ejecutables.

En las siguientes secciones se presentan las características más destacables de cada uno de los programas de MacEmulus.

5. Emulus

Con el nombre de Emulus designamos al programa MacEmulus con emulación VT-100 [8]. Ofrece al usuario de un Macintosh el acceso, mediante línea serie, a servicios que utilicen esta norma para la representación de la información. Consta de dos módulos: uno dedicado al control de las sesiones y otro de interface con Euskom, que se describe en §6. La Figura 1 muestra la barra de menú del módulo de comunicaciones.

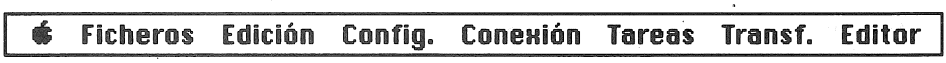


Figura 1

Para describir el programa iremos enumerando las posibilidades que ofrece cada menú.

- **Ficheros**: abrir, cerrar y salvar ficheros de configuración. Capturar parte de una sesión en un fichero de texto. Imprimir una copia de la ventana de emulación o una sección de texto de la sesión. Salir de la aplicación. Al seleccionar "Nuevo" o "Abrir" en este menú, aparece una ventana de terminal como la de la Figura 2.
- **Edición**: cortar, copiar y pegar en sesiones o en ficheros de texto.
- **Config.**: configuración de los parámetros del terminal. Características del terminal y de la línea serie, teléfono del servicio, disponibilidad o no de modem. Asignación de tareas o de cadenas de caracteres a un botón de función. Estos parámetros se pueden almacenar en un fichero de configuración.
- **Conexión**: conexión y desconexión a servicios. Util si la conexión es vía modem.
- **Tareas**: control de la ejecución de tareas (funciones automatizadas).

- **Transf.**: envío de ficheros de texto.
- **Editor**: acceso al editor de texto integrado en MacEmulus. Se pueden editar ficheros para luego ser transferidos. También se pueden editar tareas fuente y preparar a partir de ellas tareas ejecutables.

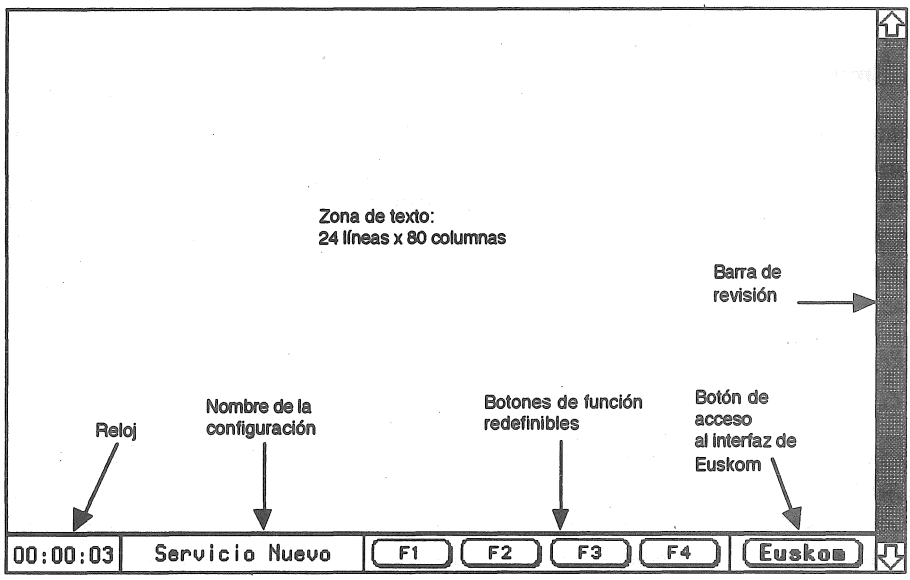


Figura 2

6. Acceso a Euskom

Al pulsar el botón "Euskom" que aparece en la parte inferior derecha de la ventana de terminal se accede al interfaz amistoso de Euskom. Normalmente se realizará esta acción una vez que hayamos accedido a este servicio. Entonces se produce un cambio en la barra de menú, apareciendo la de la Figura 3.

Figura 3

La descripción de cada menú es:

- **EUSKOM**: obtención de información sobre conferencias o usuarios. Abandono del servicio.
- **Situación**: acceso a una conferencia o al buzón personal. Suscripción a conferencias.
- **Recibir**: acceso a los mensajes recibidos y almacenados en el buzón o en la conferencia activa.
- **Enviar**: envío de notas a conferencias, buzones de usuarios, télex o máquinas fax.

- **Directorios**: gestión de directorios personales de usuarios, conferencias, télex y fax. Un directorio permite almacenar un alias junto con una dirección, ofreciendo un medio sencillo de recordar una dirección de un usuario o conferencia.

Por medio de estos menús podemos utilizar todas las posibilidades de Euskom, sin necesidad de usar la -un tanto farragosa- sintaxis de su lenguaje de comandos.

7. Emutel

Emutel es el nombre de la emulación CEPT2 (Minitel) [9] de MacEmulus. La Figura 4 muestra la barra de menú principal. Las opciones de Edición, Config., Conexión y Tareas son prácticamente las mismas que en Emulus, no existiendo el editor integrado ni la posibilidad de transmitir ficheros de texto. Las tareas ejecutables para Emutel deben prepararse previamente con Emulus.

Figura 4

La Figura 5 muestra el aspecto de la ventana de terminal de Emutel. Además de los botones redefibles ya presentes en Emulus, se han añadido una serie de botones con funciones específicas: un teclado numérico, un teclado auxiliar con funciones presentes en los terminales Minitel, un botón de captura de la pantalla y un botón de revisión.

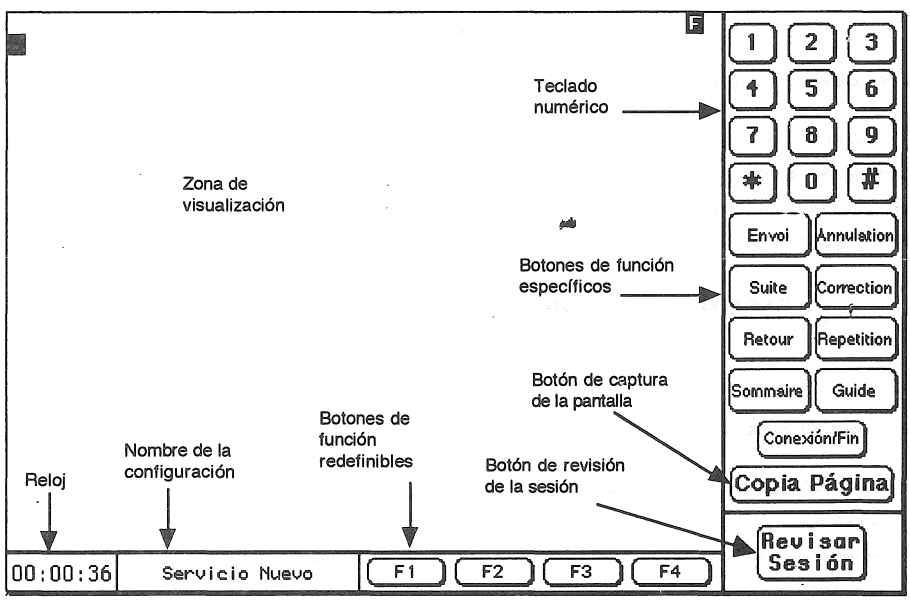


Figura 5

Una de las posibilidades más interesantes de Emutel es la posibilidad de revisar sesiones. A través del boton señalado en la Figura 5, se accede al modo de revisión, en el que es posible

avanzar página a página, hacia atrás o hacia adelante, o incluso ir directamente a una página específica. Emutel también puede almacenar las sesiones en ficheros que pueden ser más tarde revisados off-line.

Las pantallas capturadas son elementos gráficos que pueden ser integrados en otras aplicaciones Macintosh. También es posible capturar secciones de una pantalla, sin más que seleccionarlas con el ratón.

8. Emutex

Emutex es la versión de MacEmulus para emulación de terminal CEPT1 [10]. Este estándar de representación es mucho más rico en capacidad gráfica que el CEPT2 (Minitel), y es el elegido por Telefónica para Ibertex. Salvo por la diferente emulación que realiza, este programa se comporta de la misma forma que Emutel. Sólo se percibe el cambio en el diferente contenido de los botones no redefinibles, que quedan como muestra la Figura 6.

Figura 6

9. Conclusiones

De lo expuesto en los apartados anteriores, se puede concluir que el uso de MacEmulus supone un buen número de ventajas para utilizar una red de servicios telemáticos como Spritel:

- Varias emulaciones con una sola máquina, con lo que esto supone en economía y en comodidad
- Entorno común para las tres emulaciones, lo que facilita el aprendizaje y el uso de los programas
- Interfaz de usuario amistoso en línea con todas las aplicaciones Macintosh. Integración con todas las aplicaciones Macintosh en cuanto a la exportación de texto y gráficos
- Posibilidad de revisión de sesiones, lo que permite estar conectado menos tiempo a un servicio y ahorrar costes

- Transferencia de textos. Es posible preparar un documento off-line y transmitirlo rápidamente, en vez de introducirlo por teclado
- Acceso automatizado a servicios, simplificando el uso de los mismos y contribuyendo aún más a consumir menos tiempo de conexión

10. Perspectivas futuras

Hasta ahora se ha supuesto que el usuario de MacEmulus disponía en exclusiva un modem conectado a su línea serie. Siendo esta una configuración habitual, existen otras posibilidades. Por ejemplo, en la Facultad de Informática de la UPV/EHU es posible establecer una sesión LAT o Telnet desde un Macintosh equipado con una tarjeta Ethernet y un computador VAX (a través de la red local). Una vez en el VAX se puede acceder a Spritel a través de Iberpac. Pues bien, como MacEmulus está preparado para operar a través de la línea serie, no es posible utilizarlo en esta configuración.

Nuestro plan de trabajo es ampliar la potencia de MacEmulus flexibilizando la forma de conexión: aparte de la conexión por línea serie, pensamos en conexiones LAT y Telnet. Además tenemos intención de incorporar algunos protocolos de transferencia de ficheros, por línea serie (Kermit, Xmodem) o usando TCP/IP (FTP).

Por otra parte pensamos en la realización de interfaces amigables para otras aplicaciones, en la línea del que se ha realizado para Euskom. De cara al diseño de los mismos, se pretende crear un sistema de descripción de interfaces que nos permitiría realizar esta tarea de una forma semi-automatica.

Bibliografía

[1] Spritel. "Red Spritel. Puesta en marcha" Documento Spritel 14.
[2] Spritel. "Euskom. Servicios Euskom" Documento Spritel. Versión 3.0. Enero 1991.
[3] Spritel. "Euskom. Manual del usuario" Documento Spritel 15.
[4] Apple Computer. "Programmer´s Introduction to the Macintosh® Family". Addison-Wesley, 1988.
[5] Apple Computer. "Inside Macintosh" Volúmentes I, II, III, IV, V, X-Ref. Addison-Wesley.
[6] Mainstay. "V.I.P. Visual Interactive Programming". 1987.
[7] Symantec. "THINK´s LightspeedC. User's manual". 1988.
[8] Digital Equipment Corporation. "VT101 Video terminal user guide". 1981.
[9] Télétel. "Spécifications Techniques d'Utilisation de Minitel 1B". 1986.
[10] Telefónica. "Terminal Ibertex (B)".

AVALIAÇÃO DOS MÉTODOS DE AQUISIÇÃO DE CONHECIMENTO NO PROJETO GEOXPERT

Mara Abel[1],
Antonio Carlos da Rocha Costa[1] e
Jorge Carlos Della Favera[2]

(1) Instituto de Informática - Universidade Federal do Rio Grande do Sul (UFRGS) - Av.Bento Gonçalves,9500. Bloco IV. CEP 91501 - Porto Alegre, RS, Brasil - FAX 055-051-3365576
marabel@inf.ufrgs.br

(2) Centro de Ensino e Pesquisa Leopoldo Miguez de Mello (CENPES), PETROBRAS. Rio de Janeiro, RJ, Brasil

ABSTRACT

This paper discusses the elicitation process of geological knowledge for an expert system in a very imperfected domain: petroleum exploration. GEOXPERT project was developed since 1985 as a co-operation between Universidade Federal do Rio Grande do Sul (UFRGS) and the research center of Petroleo do Brasil S.A. (CENPES- PETROBRAS). GEOXPERT systems intended to provide consultational advice on both identification and classification of an specific geological environment. Factual and heuristic knowledge was provided by a sedimentary geologist considered the expert in sedimentary environment on Brazil, in an empirical process added by a prototype system , meetings and field works. The experiment and after studies resulted in a set of acquisition and representation techniques. Elicitation techniques include bibliographic review, structured and non-strutured interview and semantic analysis.

1. INTRODUÇÃO

O objetivo da pesquisa no Projeto GEOXPERT foi o domínio das técnicas de aquisição e representação de conhecimento em domínios extremamente mal-estruturados, como é o caso da avaliação de ambientes sedimentares para exploração de petróleo. O produto do projeto é um sistema especialista de classificação para identificação de rochas turbidíticas (a principal rocha-reservatório de petróleo do Brasil), a partir de dados de geologia de superfície. Os dados tratados pelo sistema são fundamentalmente simbólicos, incompletos e resultantes de interpretação subjetiva.

2. ETAPAS NA AQUISIÇÃO DE CONHECIMENTO

A aquisição de conhecimento desenvolveu-se como quatro grandes fases:

2.1 Reconhecimento do domínio e delimitação do problema.

Foram realizadas entrevistas com o especialista e os usuários do sistema e delimitados os objetivos a serem atingidos. Parte do modelo de turbiditos estudado na literatura foi implementado utilizando a estrutura de controle do sistema MicroProspector [4] como protótipo. A utilização desse protótipo simplificado foi de extrema importância em termos do diálogo e motivação do especialista.

2.2 Definição do modelo geológico formal

Consistiu de uma longa fase de estudo da bibliografia específica sobre exploração de petróleo. A seguir foram feitas uma séria de entrevistas *ad hoc* com o especialista.

A maioria dos problemas na construção do banco de conhecimentos vieram da falta de uma metodologia objetiva para aquisição de conhecimentos nesta fase de construção do sistema. A experiência apontou os principais erros cometidos que, infelizmente, não são privilégio deste projeto específico.

Em princípio, houve uma excessiva preocupação em representar de uma forma computacional o modelo de turbiditos selecionado e que estava fartamente descrito na literatura. O modelo, descrito em detalhes e exaustivamente explicado pelo especialista, é de utilidade acadêmica e auxiliar no raciocínio científico e não refletia o método de solução de problema realmente utilizado pelo especialista. Embora descrevesse como método de trabalho a observação das inúmeras minúcias que o modelo exigia, AO SER CONFRONTADO EFETIVAMENTE COM O PROBLEMA, utilizava de atalhos de raciocínio, dados diagnósticos e poucas dicas para rapidamente apresentar a solução do problema.

2.3 Detalhamento da estrutura de representação e inclusão do conhecimento heurístico.

Nesta fase, parte do sistema foi reimplementado de forma a reduzir os caminhos da inferência e permitir o tratamento de tipos específicos de informações. Entre outras feições, foram

incluídas gravuras nas descrições dos objetos, associados *índices de significância* da presença dos objetos em relação a hipótese sob avaliação, implementados procedimentos para detecção de informações contraditórias fornecidas pelo usuário e definidos os cálculos e índices de incerteza das conclusões oferecidas pelo sistema.

2.4 Avaliação e teste do sistema

Foram fornecidos dados de depósitos turbidíticos contidos nos arquivos da empresa e verificado o ajuste das conclusões através da adequação dos índices de significância associados aos objetos.

Na fase final, foi feita uma bateria de testes utilizando descrições de depósitos turbidíticos obtidos de outras fontes, inclusive com métodos de trabalho e de descrição diferentes dos utilizados pela PETROBRAS.

3. PROPOSTA DO MÉTODO DE TRABALHO

A partir do experimento do Projeto GEOXPERT e estudos posteriores [3] [5] pode-se formalizar uma proposta de um método genérico de aquisição de conhecimento, para domínios mal-estuturados e simbólicos como aqueles das Ciências Naturais (Geologia, Biologia, Meteorologia) ou Medicina. A obtenção do conhecimento deve desenvolver-se segundo as seguintes etapas:

3.1 Imersão na literatura

A fase de imersão na literatura é tanto mais necessária quanto mais desconhecido for o domínio da informação para a equipe de aquisição de conhecimento. Deve-se ter em mente que o objetivo da fase de imersão não é aprender a resolver o problema, mas familiarizar-se com os métodos e vocabulário específico da área. Deve ser evitado o julgamento de qualquer informação obtida.

3.2 Entrevistas iniciais.

São entrevistas não-estruturadas, onde o diálogo não segue nenhuma linha rígida de desenvolvimento. As perguntas devem ser genéricas e objetivas, do tipo:

O que é o objeto do conhecimento?

Como é resolvido o problema?

O especialista deve falar livremente com o mínimo de intervenções possíveis. Tudo o que foi dito ou mostrado deve ser anotado. Quando o especialista tornar-se repetitivo ou evasivo, a entrevista será encerrada.

3.3 Análise das informações

A partir das anotações obtidas na entrevista inicial com o especialista são apontados os objetos principais da informação e evidenciados os relacionamentos (hierarquias, conjuntos de informações que aparecem juntas ou tem significado especial, relações específicas). Ainda, pode ser definido o método geral de

trabalho do especialista (análise, diagnóstico, projeto, etc). Mesmo que a informação como um todo não seja compreendida pela equipe, ainda assim os *objetos* podem ser evidenciados (normalmente são os substantivos repetidamente utilizados pelo especialista), seus atributos (da mesma forma para os adjetivos) e relacionamentos (descritos como verbos).

Devem ser construídas listas de objetos reconhecidos e propostas as formas de classificação que puderem ser inferidas.

3.4 Entrevistas estruturadas

As entrevistas são feitas de forma dirigida, para evidenciar detalhes específicos das informações.

3.4.1 Evidenciando os métodos

Observar o especialista no momento em que ele realiza a tarefa e solicitar que ele descreva o que faz em voz alta (essa técnica é chamada "protocolo verbal concorrente" e é descrita, juntamente com outras técnicas de aquisição de conhecimento em [5]). Essa prática tem melhores resultados do que a entrevista não-estruturada, porque neste caso pode-se observar o que o especialista efetivamente *faz* e onão o que ele *diz que faz* ou o que ele acha que é certo fazer.

3.4.2 Evidenciando as estruturas:

Fornece-se para o especialista as listas de objetos obtidas na análise do conhecimento, solicitando que os objetos sejam sucessivamente divididos em classes sob diversos critérios, ordenados e organizados de formas diversas.

3.5 Reconhecendo as incertezas

Uma forma de evidenciar as incertezas do conhecimento é através de proposição e restrições de tarefas. Após o especialista realizar uma determinada tarefa, propor a ele formas diferentes de resolver o mesmo problema sob contexto cada vez mais restrito. Por exemplo, solicitando o que ele faria se determinada informação não estivesse disponível e com que segurança ele proporia a solução. Restringe-se cada vez mais as informações até que o especialista declare que não é mais possível resolver o problema.

3.6 Proposta de uma forma de representação de conhecimento e prototipação

Com os dados obtidos até aqui fica evidente a estrutura em que o conhecimento é organizado e qual o método do especialista, permitindo selecionar com segurança a forma de representação de conhecimento adequada. A partir dela, pode-se propor a utilização de uma ferramenta para a prototipação inicial ou mesmo para ser utilizada como ambiente final de execução. A aquisição de novos conhecimentos para a expansão do banco deve ser feita voltando às etapas 4 e 5.

4 CONCLUSÕES

A experiência com o Projeto GEOXPERT demonstrou a necessidade da utilização de métodos sistemáticos para a aquisição e formalização de conhecimento para a contrução de sistemas especialistas. A utilização de processos empíricos e estrevistas informais resulta em um conflito entre a necessidade

de *aprendizado* ou de *apreensão* da informação. O aprendizado constitui em um processo demorado, penoso e o produto tende a ser distinto do conhecimento fornecido originalmente. A aquisição de conhecimento deve ser feita através de processos impessoais e sistemáticos onde a informação do especialista não sofra julgamentos ou filtros durante sua representação.

O método aqui proposto pode ser utilizado como um ponto de partida na seleção das técnicas de aquisição de conhecimento aplicáveis a domínios simbólicos excessivamente descritivos e mal-estruturados.

6 BIBLIOGRAFIA RECOMENDADA

[1] ABEL, Mara. Um protótipo de sistema especialista para identificação e classificação de turbiditos. Dissertação de Mestrado. Porto Alegre, PGCC-UFRGS, 1988.

[2] HAYES-ROTH, F. (ed) Building expert systems. California, Addison-Wesley, 1983.

[3] LEAO, Beatriz F. Construção da base de conhecimento de um sistema Especialista de apoio ao diagnóstico de cardiopatias congênitas. Tese de Doutorado. São Paulo, PGC-Esc. Paulista de Medicina, 1988.

[4] McCAMMON, Richard B. Operation manual for the muProspector consultant system. U.S. Dept. of the Interior - Geological Survey, 1983. Open-File Report 83-804.

[5] WRIGHT, George e AYTON, Peter. Eliciting and modelling expert knowledge. Decision Support Systems 3 (1987)13-26. Elsevier Science Publishers, North-Holland, 1987.

Um Ambiente de Suporte ao Trabalho Cooperativo

Carlos A. T. Aguiar Ana Carolina Salgado

Departamento de Informática
Universidade Federal de Pernambuco
Cx. Postal 7851, 50732-970 Recife - PE - Brasil
e-mails: cat@di.ufpe.br & acs@di.ufpe.br

Abstract

The use of computer to intermediate and facilitate human interaction has increased at present. Tools that support several people working together have been defined to assist this workgroups in colaboration, communication and coordenation of their colective tasks. The Cooperative Environment proposed here, based in a simplified cooperation model, assist not only a specific activity, but a wide range of cooperative work applications.

Keywords

Computer-Supported Cooperative Work, Groupware, Cooperação, Colaboração, Comunicação, Coordenação, Modelo de Cooperação, Versões.

1 Introdução

Atividades coletivas são caracterizadas pela divisão de tarefas e troca de informações entre os participantes. Dentro deste escopo, a cooperação entre os integrantes do grupo e o compartilhamento de dados entre eles assumem um papel de extrema importância.

Computer-Supported Cooperative Work (CSCW) ou Computação Colaborativa, um dos mais novos tópicos em computação, focaliza o papel do computador na cooperação, no compartilhamento da informação, no trabalho em grupo [02]. CSCW fez surgir uma nova classe de aplicações: Groupware.

Um ambiente de suporte ao trabalho cooperativo reúne as ferramentas de software que auxiliarão os usuários (integrantes de um grupo) na realização das ações cooperativas (projeto), assistindo-os na comunicação, na colaboração (manipulação e compartilhamento da informação) e na coordenação de suas atividades (gerência do desenvolvimento do projeto).

Problemas surgem quando verificamos que os requisitos de suporte à cooperação (comunicação e colaboração entre os usuários, coordenação, controle de versões, etc) não são adequadamente atendidos até mesmo pelos ditos Sistemas de Banco de Dados não-convencionais [13]. Como consequência, *Modelos de Cooperação* têm sido definidos ([11, 13, 15, 05]) para atender às exigências do trabalho cooperativo em ambientes de desenvolvimento.

Propomos aqui um Ambiente de Suporte ao Trabalho Cooperativo que está fundamentado em um modelo de cooperação simplificado e que busca satisfazer não só às necessidades

de uma aplicação específica (CAD, por exemplo) mas de várias atividades de natureza coletiva.

O Processo de Cooperação, que inclui a definição do projeto, o mecanismo de versões e o modelo de cooperação é apresentado na próxima seção. Na sequência, descrevemos o Ambiente Cooperativo, identificando seus principais componentes. Por fim, concluimos com algumas considerações.

2 O Processo de Cooperação

Colaboração é um sinal de maturidade [10]. Uma idéia é primeiro desenvolvida para ser, então, fornecida à comunidade. A existência de ferramentas de grupo não obscurece a necessidade de aplicações individuais. O desenvolvimento individual persiste, só que atingimos um nível onde a divisão de tarefas (ou objetivos) tornar-se-á essencial para a obtenção de um resultado satisfatório e em tempo hábil.

O processo de cooperação se inicia, exatamente, nesta divisão do projeto em subprojetos e na atribuição de papéis (ou funções) aos seus integrantes. Para suportar esta divisão de tarefas e a consequente interação entre vários usuários, temos que definir um *Modelo de Cooperação*. Este, por sua vez, está fundamentado em um outro modelo, o *Modelo de Versões*.

2.1 A Definição do Projeto

Modelamos a aplicação de projeto como uma hierarquia de (sub)projetos em forma de árvore ([15, 16, 17]). Os nodos da árvore representam as tarefas a serem realizadas no âmbito do projeto, enquanto os ramos especificam as relações de dependência do tipo tarefa/subtarefa (ou objeto/subobjeto). A divisão do projeto em um conjunto de subprojetos é repetida até que os subprojetos atinjam o nível de complexidade desejado.

Cada usuário pertencente ao projeto tem pelo menos um papel (ou função) associado [08]. Identificamos quatro possíveis funções dentro do projeto: **coordenador**, **projetista**, **subcoordenador** e **revisor**. Todo projeto tem um coordenador (normalmente aquele que sugere a definição de um projeto) e vários projetistas. A figura do subcoordenador só aparece quando um subprojeto é novamente dividido. A este subprojeto associamos um subcoordenador. Já os projetistas estão relacionados aos subprojetos terminais da árvore. O revisor tem a função específica de comentar documentos já liberados (versões não-privativas), sugerindo modificações aos seus autores.

À parte da estrutura hierárquica do projeto, definimos uma ordem parcial de desenvolvimento para os subprojetos, estabelecendo uma dependência temporal entre eles. Além disso, cada subprojeto identificado tem um *deadline* para conclusão e um conjunto de objetivos a serem cumpridos.

Podemos mostrar, através de um exemplo, como seria definido um projeto. Este exemplo não pretende ser completo, mas deve ser o suficiente para se ter noção do processo de definição de um projeto.

O projeto-exemplo envolve a aquisição e análise de sinais EEG (eletroencefalográficos) [01]. Em síntese o sistema Tecnolab compreende:

- Conversão analógico-digital, que transforma uma variação de tensão em um sinal digital;
- Aquisição e amostragem (visualização) do sinal digital, em tempo-real;
- Análise do sinal adquirido em um determinado intervalo de tempo.

O objetivo do projeto é, em linhas gerais, pegar o sinal analógico, convertê-lo para digital (de forma que o computador possa tratá-lo), fazendo sua aquisição e amostragem durante um certo intervalo de tempo para posterior análise.

Uma primeira divisão em subprojetos identifica três tarefas (ou objetivos), que correspondem aos itens anteriormente especificados, ou seja, conversão, aquisição/amostragem e análise.

Apesar da aquisição/amostragem estarem reunidas em um único subprojeto, pois estão intimamente relacionadas, podemos pensar em mais um nível de detalhamento, dividindo esta tarefa em duas subtarefas:

- aquisição do sinal digital;
- amostragem (visualização) deste sinal;

Deste modo, podemos pensar em um módulo de amostragem que recebe um valor qualquer e transforma-o em um ponto gráfico. Não sendo necessário qualquer outro desdobramento, chegamos a hierarquia de projetos mostrada na figura 1.

De acordo com a hierarquia de subprojetos, associamos e definimos os papéis dos integrantes do grupo (como pode ser visto na figura 1):

- Um **coordenador** tem a função de coordenar o projeto como um todo;
- Ao subprojeto *conversão A/D* podemos associar um ou mais **projetistas**;
- Como o subprojeto *aquisição/amostragem* foi subdividido, definimos a figura do **subcoordenador** neste nível, e para cada subprojeto deste associamos um ou mais **projetistas**;
- À *análise* também podemos associar um ou mais **projetistas**.

Os objetivos são descritos informalmente, apesar de aplicações de teste e validação poderem ser utilizadas sobre as versões produzidas. Por exemplo, para este sistema ser viável, devemos verificar se o tempo gasto na amostragem de um ponto não é maior que o tempo entre duas aquisições sucessivas. Isto pode ser determinado verificando o tempo gasto pelo módulo de visualização para amostrar um ponto. Se esta restrição não estiver garantida, o objetivo ainda não terá sido atingido.

Definido o projeto, partimos para o seu desenvolvimento. Todo o trabalho envolvido é realizado sobre versões de objetos, as quais são resultado dos vários subprojetos. Para suportar o conceito de versões, definimos um *Modelo de Versões*.

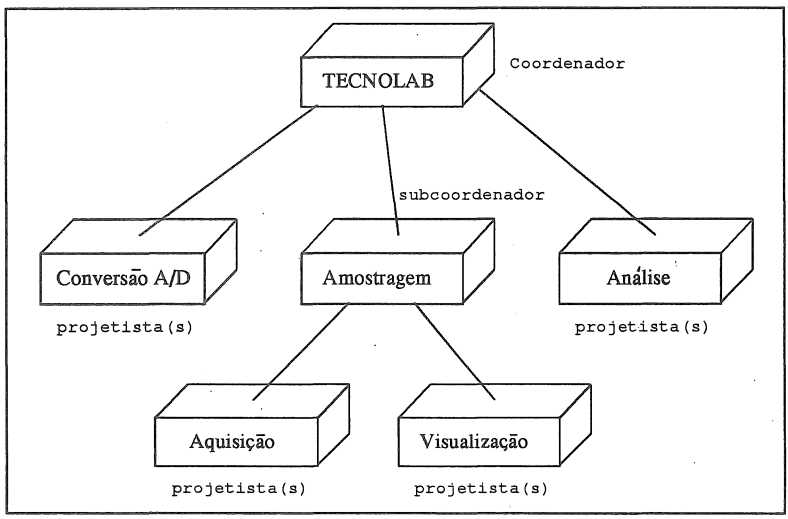


Figura 1: Hierarquia de projetos do Tecnolab

2.2 Modelo de Versões

Durante o desenvolvimento de um projeto, os subprojetos deste passam por vários estágios de implementação, gerando o que chamamos de versões de objeto. Depois da criação inicial do(s) objeto(s) do subprojeto, novas versões deste objeto podem ser derivadas, e outras podem ser derivadas destas, formando uma hierarquia de derivação de versões para o objeto. Uma hierarquia de derivação de versões captura a evolução do desenvolvimento e indica a ordem parcial das versões do objeto [04].

Em nosso modelo, identificamos três tipos de versões, em relação a sua disponibiliade e aos tipos de operação que podem ser realizadas sobre elas: *versões privativas*, *versões de (sub)projeto* e *versões públicas* [14].

Uma versão privativa tem as seguintes propriedades:

- ela só pode ser consultada e atualizada (inclusive eliminada) pelo usuário que a criou; quando mais de um projetista está associado a um mesmo subprojeto haverá mais de um autor, e ai teremos o compartilhamento da versão privativa;

- uma nova versão privativa pode ser derivada de uma outra versão privativa já existente. A versão existente pode, então, ser promovida a versão de (sub)projeto;

- ela é armazenada em uma área acessível apenas ao usuário (ou usuários) que a criou (banco de dados privativo).

Versões de (sub)projeto têm as seguintes características:

- ela pode existir tanto a nível de projeto como a nível de subprojeto;
- ela não pode ser atualizada;
- ela pode ser removida pelo seu criador;
- uma versão privativa pode ser derivada de uma versão de (sub)projeto;
- uma versão de subprojeto é gerada a partir da promoção (explícita ou implícita) de uma versão privativa; uma versão de projeto pode ser criada a partir de uma versão privativa ou de subprojeto.

Versões acessíveis a todos os usuários, de qualquer projeto, são versões públicas. Como versões de (sub)projeto, elas não podem ser atualizadas, podem ser removidas pelos seus criadores e podem originar uma versão privativa.

Uma versão privativa pertence ao usuário que a criou, e a sua existência não será de conhecimento dos outros usuários do projeto até que ela seja promovida a versão de (sub)projeto, liberada pelo seu criador e avalizada pelo (sub)coordenador. Uma versão de subprojeto é definida quando um subprojeto não está diretamente ligado à raiz da hierarquia de (sub)projetos. O subprojeto que originou esta divisão agrupa tais versões. Tomando como exemplo o projeto definido anteriormente (figura 1), teremos versões de subprojeto associadas ao módulo *aquisição/amostragem*, originadas dos subprojetos *aquisição* e *amostragem*.

Tendo à disposição uma versão de (sub)projeto, todos os usuários associados a subprojetos derivados na hierarquia podem utilizá-la como tal, mas apenas com permissão de leitura, sem possibilidades de atualização. No entanto, como identificado anteriormente, versões privativas poderão ser geradas a partir destas e, portanto, atualizadas sem qualquer alteração nas versões de (sub)projeto que as originaram.

2.3 Modelo de Cooperação

Modelos de cooperação têm sido propostos ([11, 13, 15, 05]) não só para suprir a falta de controle automático de consistência no acesso concorrente ao banco de dados, gerada pela falta de isolamento e atomicidade das transações cooperativas, mas também com o objetivo de definir novos critérios tanto para o controle da cooperação entre projetistas (usuários de um grupo) como para a gerência do desenvolvimento do projeto como um todo [12].

Em nosso modelo, projetistas interagem através da utilização de versões de objetos disponíveis a todos usuários associados a subprojetos derivados na hierarquia (versões de (sub)projeto). Ao interagirem através de uma versão de (sub)projeto, os projetistas assumem ou o papel de criador ou o de usuário da versão. Ao usuário da versão cabe o direito de utilizá-la, mas sem possibilidades de alteração; ao criador, a condição de removê-la, mas só quando esta não estiver sendo usada por qualquer outro projetista.

Nosso modelo prevê o compartilhamento de áreas privativas, quando mais de um projetista esta engajado em um mesmo subprojeto. Neste caso, um dos autores bloqueia uma

versão de projeto quando passa a utilizá-la. Este bloqueio é mantido durante toda a atividade de atualização, podendo ser retirado após um período que indique a falta de uso (*idle period*). Os outros usuários que compartilham a área serão notificados do bloqueio desta versão.

Além disso, o modelo permite, através da estrutura hierárquica de derivação de versões, que o processo evolucionário da construção dos objetos de projeto esteja disponível aos projetistas. Assim, eles têm uma visão global do projeto, bem como de sua evolução. Estas e outras funções são atribuições do *Gerenciador de Cooperação* que mantém o estado do projeto através da manutenção e controle do estado dos seus subprojetos.

Com este enfoque, obtemos um modelo de cooperação genérico o bastante para atender às necessidades dos usuários no desenvolvimento de software, CAD, co-autoria, etc, atividades cooperativas por natureza.

3 O Ambiente de Cooperação

Ambientes voltados para o trabalho em grupo referem-se a sistemas de software que suportam dois ou mais usuários trabalhando, simultaneamente ou não, numa tarefa comum e que provê uma interface do ambiente compartilhado ([07, 06]). Neste ambiente compartilhado as ações de um usuário são visíveis aos outros integrantes da equipe de trabalho. Seu objetivo é assistir grupos de usuários na comunicação, colaboração e coordenação de suas atividades.

A existência de um espaço compartilhado não elimina a necessidade de um espaço privativo, espaço ao qual apenas o usuário tem acesso, e onde não haja restrições quanto ao uso de aplicações diversas. A liberdade de escolha do usuário deve ser mantida.

Um ambiente de suporte ao trabalho cooperativo deve fornecer subsídios para a manipulação e compartilhamento da informação (colaboração), coordenação das atividades cooperativas (gerência do desenvolvimento) e comunicação entre os participantes de um projeto. O ambiente proposto consegue suprir essas necessidades através da implementação de três módulos: o *Gerenciador da Informação*, o *Gerenciador de Cooperação* e o *Sistema de Comunicação*. Descrevemos a seguir, de forma sucinta, cada um deles.

3.1 Gerenciamento da Informação

Nosso ambiente está definido em termos de projetos e de usuários (projetistas, revisores, subcoordenadores e coordenadores) que cooperam dentro de cada projeto. Um projeto reúne, portanto, um grupo de usuários e um conjunto de objetos (que podem ser privativos, de projeto ou públicos). Os objetos (versões dele) no banco de dados de (sub)projeto estão disponíveis a todos aqueles usuários ligados ao (sub)projeto, enquanto os objetos no banco de dados privativo são de propriedade e estão acessíveis apenas a seu(s) autor(es). Um projetista pode compartilhar dados (versões de objetos) com outros projetistas do mesmo grupo através da área de (sub)projeto [04].

A hierarquia do banco de dados (BD) em nosso ambiente inclue os bancos de dados privativos, os bancos de dados de subprojeto e projeto, e o banco de dados público. Além

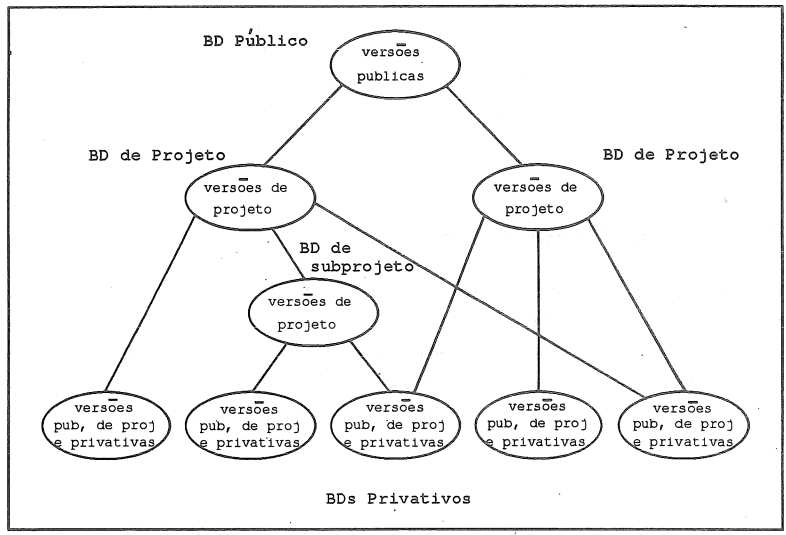


Figura 2: Hierarquia do Banco de Dados do Ambiente

destes, um banco de dados de controle mantém informações relevantes ao ambiente, mas não faz parte desta hierarquia. A figura 2 representa um exemplo de hierarquia.

Versões de (sub)projeto podem existir tanto no BD de (sub)projeto como no BD privativo; já versões privativas só estão definidas nos BDs privativos. Os BDs privativos são compartilhados por mais de um projeto quando o usuário integra mais de um grupo de projeto; são compartilhados por mais de um projetista (apenas a parte comum ao subprojeto) quando estes estão associados ao mesmo subprojeto.

Bancos de dados privativos estão armazenados na área do usuário, enquanto os de (sub)projeto estão distribuídos em uma rede local de forma a otimizar o acesso à informação por parte dos projetistas.

3.2 Gerenciamento da Cooperação - a interface do espaço compartilhado

O Gerenciador de Cooperação é o responsável pela manutenção do estado atual do projeto, permitindo que ações de um usuário sejam visíveis aos outros projetistas do grupo [12]. Entendemos como *ações visíveis* aquelas que afetam o (sub)projeto em questão, mas que não fazem parte do espaço privativo do usuário.

O usuário desenvolve sua tarefa dentro de uma *sessão de trabalho*. Nesta sessão o usuário passa a ter um papel (ou função) bem definido. A atribuição de papéis (ou funções) aos usuários integrantes de um projeto permite que o gerenciador de cooperação determine

o nível de acesso à informação, a partir do controle da interface do ambiente compartilhado.

Informar os usuários sobre o estado de suas ações, das dependências e objetivos atingidos, bem como notificá-los de acordo com a necessidade, são atribuições do gerenciador de cooperação.

A interface compartilhada dá a exata noção do estado atual do (sub)projeto e fornece ao usuário as ferramentas necessárias à cooperação entre os projetistas.

3.3 Sistema de Comunicação

O Módulo de Comunicação visa suprir as necessidades básicas de comunicação entre os membros de uma equipe que realizam um trabalho conjunto [03].

Aproveitando o ambiente de cooperação criado e controlado pelo Gerenciador de Cooperação, os usuários da rede (e pertencentes a um projeto) podem trocar pequenas mensagens de forma rápida e simples. A definição de papéis permite também que usuários referenciem outros integrantes do grupo, através da função deles dentro do projeto (por exemplo, um projetista se comunica com o coordenador de projeto sem a necessidade de saber quem ele é).

O mecanismo de e-mail fornecido pelo sistema operacional está sendo adaptado e expandido para o tratamento de documentos multimídia, especificamente o que diz respeito a manipulação (criação, "visualização", armazenamento e envio) de sons, com o objetivo de tornar mais natural (e poderosa) a troca de informações entre os usuários do ambiente cooperativo. Um elemento sonoro é uma passagem de audio digitalizada, fornecida ao ambiente por um Servidor de Audio e manipulado pelo usuário através de uma interface gráfica que, funcionalmente, se assemelha a um gravador portátil.

Anotações associadas a versões de objetos são tratadas independentemente, já que versões de projeto não são atualizáveis. Através delas, projetistas podem propor e solicitar modificações de objetos aos seus criadores.

Por fim, um ambiente de conferência com protocolo preemptivo deve ser incorporado. Nele, todos os participantes têm a mesma visão da janela de comunicação (WYSIWIS [1]) e apenas um deles tem o direito de "fala" por vez [03].

4 Conclusão

Uma proposta de Ambiente de Suporte ao Trabalho Cooperativo foi apresentada de forma resumida. O Ambiente tem por objetivo auxiliar grupos de pessoas trabalhando de forma cooperativa, fornecendo mecanismos de colaboração, coordenação e comunicação.

Sua simplicidade no tratamento da cooperação segue a premissa de que não é necessário (nem interessante) mudar a forma de interação entre as pessoas no trabalho em grupo; é necessário sim, entender como isso se processa e permitir a improvisação característica das atividades coletivas [09].

[1] *What You See Is What I See*

Referências

[01] C.A.T. Aguiar, F. Borges & J. Ceppola, "Sistema Automatizado para Registro e Análise de Sinais EEG", I Congresso Brasileiro de Informática em Saúde, Campinas, 1986

[02] C.A.T. Aguiar & A.C. Salgado, "Computer-Supported Cooperative Work - Uma Visão Geral", Relatório Técnico, Depto. de Informática - UFPE, Recife, 1992

[03] L.S. Castro & R. Drummond, "Sistrac: Sistema de Suporte a Trabalho Cooperativo", Dissertação de Mestrado, DCC - Unicamp, Campinas, 1991

[04] H.T. Chou & W. Kim, "A Unifying Framework for Versions in a CAD Environment", Proceedings of the 12th International Conference on VLDB, Kyoto, Japan, 1986

[05] M. Dürr & K. Rösch, "A Model for Cooperative Transactions", VII Simpósio Brasileiro de Banco de Dados, Porto Alegre, 1992

[06] C.A. Ellis, S.J. Gibbs & G.L. Rein, "Groupware: Some Issues and Experiences", Communications of the ACM, Vol 34, Number 1, January 1991

[07] S.J. Gibbs, "LIZA: An Extensible Groupware Toolkit", CHI'89 Proceedings, Austin, 1989

[08] I. Greif & S. Savin, "Data Sharing in Group Work", in Computer-Supported Cooperative Work: A book of Readings, Edited by Irene Greif, Morgan Kaufmann Publishers, 1988

[09] J. Grudin, "Why CSCW Applications Fail: problems in the Design and Evaluation of Organizational Interfaces", CSCW'88 Proceeding, Portland, Oregon, 1988

[10] J. Grudin, "CSCW : Introduction", Communications of the ACM, Vol 34, Number 12, December 1991

[11] C. Iochpe, et al., "Um Modelo de cooperação baseado em características de objetos e em ordens parciais de passos de projeto", Relatório de Projeto, CPGCC - UFRGS, 1991

[12] C. Iochpe, et al., "Suporte à Cooperação em Ambiente de Projeto Assistido por Computador", VI Simpósio Brasileiro de Banco de Dados, Manaus, 1991

[13] W. Käfer, "A Framework for version-based Cooperation Control", Research Report, Universität Kaiserlautern, Alemanha, 1990 (submetido a publicação)

[14] A.M.B. Martiniano, "Gerenciamento de Versões de Objetos", Relatório Técnico, Depto de Informática - UFPE, Recife, 1992

[15] M.H. Nodine, et al., "Cooperative Transaction Hierarchies", Technical Report, no. CS-90-03, Brown University, 1990

[16] M.H. Nodine, et al., "Synchronization and Recovery in Cooperative Transactions", Proceedings of the 4th Internacional WorkShop on Persistent Object System Design, Implementation and Use, 1990

[17] A.H. Skarra, "Localized Correctness Specifications for Cooperative Transactions in an Object-Oriented Database, Office Knowledge Engineering, vol. 4, 1990

A Schema Modification Methodology for an Object-Oriented Database System

Reda AL-HAJJ and M. Erol ARKUN
Bilkent University
Faculty of Engineering and Science
Department of Computer Engineering
and Information Sciences
06533 Ankara, TURKEY

ABSTRACT: *Proper and efficient handling of schema changes is an important aspect of any object-oriented database management system. This paper deals with the definition of a schema modification methodology for ODS; an object-oriented database management system prototype designed at Bilkent University. A brief description of the basic features of ODS is included to facilitate the treatment. The invariants of the class lattice as well as the rules for resolving ambiguities due to schema changes are identified. In addition, schema changes handled by ODS are described within the realm of the specified invariants and rules. The described methodology efficiently handles deletions which are unmanagable in object-oriented systems due to the unidirectional relationship between objects in different classes.*

Keywords : *database systems, object-oriented databases, schema evolution.*

1. Introduction

Object-oriented systems evolved to satisfy the needs of such application areas as Artificial Intelligence, Computer Aided Design/Manufacturing and Office Information Systems [4, 7, 14, 19, 21], where information about the domain is incomplete or becomes available incrementally or even highly subject to change. These application areas require flexibility in changing the database schema. This necessitates that schema changes should be one of the basic characteristics to be considered in judging the power of an object-oriented database system. The flexibility in the underlying data models adds much to the object-oriented database systems. Clearly, it is desirable for an object-oriented database system to satisfy as many schema operations as possible.

W. Kim classifies in [11] schema changes among a number of architectural concepts developed for relational database systems that should be treated within the realm of object-oriented data models. Extensibility is considered in [3] as one of the characteristics that a system must possess in order to be termed an object-oriented database system.

Different data models exist each allowing a wide variety of schema changes. O2 [7] provides two modes for running an application. Schema changes are allowed in the development mode, but not in the execution mode where the schema is frozen and changes to it are forbidden. Schema changes in ORION [4, 5, 6] are based on multiple inheritance while in GemStone [16] simple inheritance based schema changes are treated. In [17, 18] the approach used is based on keeping versions to maintain a consistent view of the type lattice after a schema change.

A basic consideration in schema evolution is how to bring existing objects in line with a modified definition of an existing class. Either all instances of a modified class are instantaneously changed or they are modified only when used, otherwise remain unchanged. ORION [6] and ObServer [19] follow the approach known as screening where the change is delayed and values are either filtered or corrected as they are used. Another approach, known as conversion, is used by GemStone [16] where all instances of a class are modified in accordance with the change. The first approach sounds more sensible as there is no need to do something that will not be used.

How different items of the database are treated by the storage model affects the flexibility of schema changes. In this paper, we address the schema modification methodology defined for ODS; an object-oriented database management system prototype designed at Bilkent University. These schema changes are performed through the user interface part of ODS [21] and are achieved via the flexibility in the storage and indexing models proposed in [1, 2], respectively. Section 2 includes brief descriptions of the basic features of ODS. The invariants and the conflict resolving rules which validate schema modifications of ODS are stated in section 3. The schema evolution functions

allowed by ODS are discussed in section 4. The impact of the storage and the indexing models on schema changes are discussed in section 5. Section 6 includes the conclusions.

2. Basic Features of ODS

The implementation of ODS has been carried out using the C programming language on a UNIX workstation. In ODS, a single entity of the real world is modeled as an object. ODS supports modeling of complex objects and relationships directly so that the properties of objects are not constrained to be simple data values. Each object in ODS is identified by a unique identifier called an object-oriented pointer (OOP). Object-oriented pointers are implemented through surrogates. An object table is used to map the OOP of an object to its physical location in main memory [8].

Objects are grouped into classes. A class is allowed to inherit from more than one class, i.e., ODS supports multiple inheritance. A class contains necessary information to construct and use its instances [8]. Class variables can be added to a class definition. The root of the class lattice is the special class called OBJECT. Users are allowed to define new classes that are direct or indirect subclasses of the OBJECT class.

To preserve the encapsulation feature, ODS does not allow direct access to values of objects. Object identity is the only thing visible to the outside of an object and message sending is the only means for manipulating the properties of an object [15]. For every message there is a corresponding method. Additionally, ODS supports a user interface that includes a class browser and a programming shell as described in [21]. The storage and indexing models developed for ODS are briefly described next.

In the storage model proposed in [1], the object storage model is treated as a 3-dimensional system: the instance dimension, the nesting dimension and the inheritance dimension. The following three relations are introduced in [1]:

(x R1 y) iff x and y are instances in the same class.
(x R2 y) iff y is an immediate nested chunk of x.
(x R3 y) iff y is an immediate super chunk of x.

Information in an object is distributed between the Segment that contains atomic valued instance variables of all the chunks of one class, the Inheritance Table (IT) that contains information relating class instances to the OOPs of their immediate supers, and the Nesting Table (NT) that contains information related to the immediate nested objects, i.e., objects that form the values for non-atomic valued instance variables. Relations R1, R2 and R3 are applied to objects in the database to build the Segment, NT and IT, respectively [1]. Such a storage structure facilitates the handling of schema modifications as described in this paper and provides a fine degree of granularity [1].

Indexing is required to improve the performance of object-oriented systems. In GemStone [13] the term identity index is used where a predefined path leads to a set of index components one for each term in the path. ORION [12] uses class-hierarchy indexing and nested-attribute indexing for what we call indexing in the inheritance and nesting dimensions, respectively. On the other hand, the key step in the indexing approach proposed in [1, 2] is the recognition that two of the basic object-oriented constructs are inheritance and complex objects. Accordingly, objects in different classes should be related via these two constructs. Two types of indexes are identified; inheritance-nesting index and equality index. The inheritance-nesting index is based on the inverse of the relations R2 and R3 defined via inheritance and complex object constructs.

(y $R2^{-1}$ x) iff an object x exists with an instance variable having object y as its value.
(y $R3^{-1}$ x) iff x is a subobject of y or x inherits from y.

Two types of variable length index records are constructed. The first shows for each object the identities of its subobjects, while the second shows the identities of objects that are referencing an object as the value for their non-atomic valued instance variables. The two types of records are collected in the SubObjects Table (SOT) for the first type and the Referencing Objects Table (ROT) for the second type [1].

Concerning equality index, a B^+ tree is built to relate the value of a particular instance variable to the identities of objects in a class. Using the equality index, the identities of the desired objects are obtained. Using the inheritance-nesting index it is possible to start at any object and reach to any other related object [1, 2].

Example: The aim of this example is to facilitate the understanding of the described constructs as well as to aid in showing the handling of the different schema evolution functions described next in Section 4.

Consider a class Cj with classes Ci1, Ci2, ..., Cin found in the superclass list of Cj and classes Ck1, Ck2, ..., Ckm found at the immediate nesting level of class Cj as shown in Figure 1. All the classes found in a given class lattice are handled by the same way as class Cj, because the described relationships are only applicable at both the immediate superclasses level and immediate nesting class level.

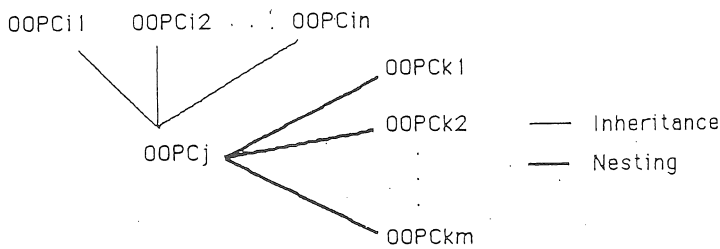


Figure 1: The general scene of a class in the class lattice

Concerning Figure 1, the information which should be included in the IT, NT, SOT and ROT is as follows:

IT(OOPCj,<OOPCi1, OOPCi2, ..., OOPCin>)
NT(OOPCj,<OOPCk1, OOPCk2, ..., OOPCkm>)
SOT(<OOPCi1, OOPCj>, <OOPCi2, OOPCj>, ..., <OOPCin, OOPCj>)
ROT(<OOPCk1, OOPCj>, <OOPCk2, OOPCj>, ..., <OOPCkn, OOPCj>)

3. Constraints of Schema Evolution and Conflicts Resolving Rules

After schema changes the class lattice should preserve some properties in order not to leave the database in an inconsistent state. This section includes the properties of the class lattice that ODS preserves upon schema modifications. In addition to the constraints there are some rules that are used to resolve conflicts resulting from schema changes.

Class Lattice Invariant

The class lattice of ODS is defined as a directed acyclic graph with a single root, the OBJECT class. It is connected, i.e., there is at least a path from each class to the root. Any schema change to the class lattice should maintain this property.

Non-empty Class Invariant

A user-defined class whose instance and class variables and methods are deleted is considered empty. An empty class only connects its subclasses to its superclasses without adding any property to the connection and hence its presence is meaningless. The class lattice should not contain any empty user-defined class. If a schema change results in an empty class, this class is automatically dropped.

Distinct Names Invariant

No two classes can have the same name. Furhter, no two methods defined within the same class

can have the same name. In addition, the instance variables within the same class should have distinct names. Name conflicts resulting from the inheritance of objects and methods are resolved according to the following two rules:

Name Conflict Resolving Rule
On name conflicts priority is given to the target class and the priority decreases while going away from the class towards the OBJECT class. For classes that have the same priority according to this rule, i.e., classes found in the same superclass list of the target class the conflict is resolved according to the order in the list. The class found at the head of the list is given the highest priority while that at the tail of the list is given the lowest priority. The OBJECT class has the least priority.

Inheritance Priority Rule
The addition of instance variables (methods) to a class and the deletion of instance variables (methods) from a class may violate conflicts that are resolved according to the above priority rule. An instance variable (method) in name conflict with another one may be inherited on deleting the latter. Further, currently inherited instance variables (methods) may cease to be considered after the addition of an instance variable (method) with the same name to a class with a higher priority.

Full Inheritance Invariant
A class inherits all the instance variables and methods from its superclass(es). No selection is done but name conflicts are resolved according to the name conflict resolving rule and the inheritance priority rule.

Homogeneous Domain Invariant
The domain for each instance variable should be bound to a specific class in the class lattice. When a class is specified as a domain, all its direct and indirect subclasses may be used in the same context because a class includes instances of its all direct and indirect subclasses.

Class Addition/Deletion Invariant
A new superclass A of an existing class B should be added as the last superclass in the order of superclasses of B. Moreover, on deleting B from the class lattice its supers should replace it as immediate superclasses of the immediate subclasses of class B. Superclasses of B are added at the end of the list because all such classes have lower priority than those classes in the same list with B. The OBJECT class should be the superclass of a newly added class unless the supers of the added class are explicitly specified.

Data Integrity Invariant
On deleting an instance from a class it is necessary to consider and drop the references to the deleted instance. No dangling reference should be kept.

Object Deletion Rule
The policy to be followed for the deletion operation is that an object can not be deleted unless it is a root object. The term root object means an object which is not an instance of any subclass; a root object has no entry in the SOT.

4. Schema Evolution Functions Allowed in ODS

In this section we described persistent schema changes allowed in our system. The description depends on the storage and indexing models of our system.

Dropping References to a Deleted Chunk
The introduction of the IT, NT, ROT and SOT facilitates much to the maintenance of database integrity. These tables help in dropping all the references to a deleted chunk in accordance with the data integrity invariant and without violating the object deletion rule. Via the ROT and SOT references to a deleted chunk can be efficiently dropped without any trouble. For instance on deleting Ci2 in Figure 1 from the class lattice, the reference from Cj to Ci2 is dropped inside the IT after being reached using the information in the SOT.

Adding an Instance to a Class
An instance is added to an existing class by adjusting the IT, NT, ROT, and SOT to reflect the

relation between the new instance and other related instances existing in the database. The new instance uses all the facilities available to other instances in its class.

Adding an Instance Variable to a Class

The addition of a new instance variable to a class can be done in two different ways each depending on whether the instance variable is atomic or non-atomic. In the first method, applicable when the new instance variable has an atomic value, the segment of the class to which the new instance variable is to be added, is accessed. The segment's contents are changed by adding the value of the new instance variable to each chunk in the segment. In the second method, applicable when the new instance variable has a non-atomic value, the NT is adjusted to reflect the relationship between instances of a class and the domain of the new added instance variable. According to the distinct names invariant, the new instance variable should have a name distinct from all variables in its class. Further, the addition of a new instance variable may affect the inheritance priority according to the name conflict resolving rule and the inheritance priority rule.

Changing the Domain of an Instance Variable

The domain of an existing instance variable can be changed as one of the following four choices: 1) from atomic domain to atomic domain; 2) from atomic domain to non-atomic domain; 3) from non-atomic domain to atomic domain; 4) from non-atomic domain to non-atomic domain. The first choice does not cause any trouble; but for the other three choices which contain a non-atomic domain the relationships are maintained in the NT and ROT to reflect the change.

Changing the Name of an Instance Variable

The name of an existing instance variable may be changed without any trouble. This is because data encapsulation is one of the distinguishing features of ODS and instance variables are treated by message passing. Such a schema change may affect the inheritance priority according to the inheritance priority rule and the name conflicting rule. Furthermore, the distinct names invariant should be satisfied.

Dropping an Instance Variable from a Class

It is possible to drop an existing instance variable from a class. The deletion of an instance variable depends on whether the instance variable has a chunk (or a collection) as its value or just an atomic value. An instance variable which has a chunk (or a collection) as its value can be deleted from the NT by deleting the information that represent the value of the instance variable from the record of the chunk. The ROT is adjusted to reflect the change. The deletion of atomic valued instance variables can be done within the related segment. Inheritance priority may change according to the name conflict resolving rule and the inheritance priority rule. Further, the non-empty class invariant should continue to hold after the deletion.

Adding a Class to the Superclass List of an Existing Class

An existing class may be added to the superclass list of another class without violating the class addition/deletion invariant. The addition is done at the tail of a superclass list. For instance an existing class, say Cir, may be appended at the tail of the superclass list of class Cj in Figure 1. Class Cir will have the lowest priority in the list.

Deleting a Class from the Superclass List of an Existing Class

A class C may be dropped from the superclass list of another class D. The super(s) of class C will replace it in the superclass(list) of its subclass D as immediate super(s). The SOT is adjusted to drop the reference(s) that existed prior to the deletion of class C. For instance on deleting Ci2 from the superclass list of Cj in Figure 1, classes found in the superclass list of Ci2 replace Ci2 in the superclass list of Cj.

Changing the Priority of the Superclasses of an Existing Class

The order of the superclasses of a certain class may be rearranged within the IT that shows for each class its immediate superclasses. For instance classes Ci2 and Cin may interchange their places within the superclass list of class Cj and hence the priority of Cin becomes higher than the priority of Ci2. Modifying the superclass list helps in changing the priority of the superclasses found at the same level with respect to a target class. These schema changes affect the inheritance priority maintained by both the name conflict resolving rule and the inheritance priority rule.

The Addition of a New Class

A new class may either have the OBJECT class as a superclass or else other existing classes in its superclass list. Furthermore, a new class may have zero,one or more subclasses. The non-empty class constraint and class lattice invariant should be maintained. According to the distinct names invariant, the name of the new class should not match with any of the existing classes. Chunks in the new class, being related by R1, are put in a new Segment. According to the full inheritance invariant, the new class inherits all instance variables and methods from its superclass(es). Instances in the new class are related to instances in other classes either by the relation R2 or R3 and hence $R2^{-1}$ and $R3^{-1}$. These relations are reflected by adding entries to the IT, NT, ROT and SOT to reflect the position of the instances of the new class with respect to existing instances of other classes. For instance considering class Cj in Figure 1 as an addition of a new class with the specified classes along both the inheritance and nesting dimensions as shown in Figure 1; the information which should be reflected into each of the IT, NT, ROT and SOT is the same information described in the example given in section 2.

Dropping a Class

The instances, instance variables and methods of an existing class C may be dropped. If all the definition and contents of class C are dropped, then class C should be deleted as an empty class to preserve the non-empty class invariant. The immediate supers of class C replace it in the inheritance mechanism for not to leave its subclasses dangling and violate the class lattice invariant. After the deletion, the inheritance priority may change according to the inheritance priority rule and the name conflict resolving rule.

Changing the Name of an Existing Class

The name of an existing class may be changed, but without violating the distinct name invariant. So, the new name assigned to the class should be different from those of other existing classes.

Adding a Method to a Class

It is possible to add a method to the definition of an existing class. But distinct names invariant should be preserved by forcing the name of the new method to be distinct from those existing in the same class. Moreover, inheritance priority may change following both the inheritance priority rule and the name conflict resolving rule.

Dropping a Method from a Class

An existing method may be dropped from the definition of an existing class. A method may be deleted after becoming irrelevant due to schema changes. A deleted method may be replaced by a new method added to the class definition or an existing method in another class according to both the inheritance priority rule and the name conflict resolving rule.

Finally, it is important to state that the algorithms used to handle the described schema modifications are found in [1].

5. The Impact of the Storage and Indexing Models on Schema Evolution Functions

An important advantage due to the separation of non-atomic references through IT, NT, ROT and SOT is realized here, whereby instance independence, nesting independence and inheritance independence are achieved. The IT, NT, ROT and SOT contain all the associations between chunks which are instances in different classes. Changes to an edge in the class lattice can be performed solely via the IT without any need to access individual chunks. Making a class a superclass of another class can be done solely via the IT that contains information related to the change. Removing a class from the superclass list of a class can again be done solely via the IT that contains information related to the change by dropping references to instances of the deleted class. Changing the order of the superclasses of a class can also be accomplished solely within the IT. By this way, inheritance independence is achieved. On the other hand, a schema change that involves a non-atomic valued instance variable is done inside the NT and the ROT is adjusted to reflect the change and so, nesting independence is achieved.

A new atomic valued instance variable may be added to a class definition requiring that the values of the new instance variable be added to the existing chunks found in the segment representing the

class. So the instance indepndence is achieved.

5.1 Schema Changes and Indexing

It is important to consider the relationship between schema changes and the indexing model. We claim that schema changes result only in minor changes to existing indexes, namely to the records of the classes that are directly involved in the change. Queries that use these indexes are not affected because a query that has recognized a change is instantiated in a forward and backward fashion. How and to what extent this claim is achieved is discussed next.

The identity index introduces SOT and ROT that show the immediate nesting and the immediate referencing objects for all objects found in the database. The effect of a schema change does not propagate further than the immediate neighbors of the changed item. A schema change will at most affect the immediate neighbor objects of the object that undergoes the change. For instance the addition of a new class between classes Cj and Ci2 in Figure 1, only affect entries related to Cj and Ci2 in the IT. The same thing is true for the addition of a class along the nesting dimension. So the addition of a class to the class lattice affects only the immediate subclasses, immediate nested classes, and immediate referencing classes and nothing more.

The identity indexes will not be affected by the addition of new instances to a class or the deletion of existing instances from a class where entries need to be added to or deleted from the SOT, ROT, IT and NT. For instance, suppose that instances in class 'student' reference instances in class 'person' and instances in class 'teacher' along the inheritance and nesting dimensions respectively. An identity index is automatically set up for instances of classes 'student', 'teacher', and 'person'. When the class 'course' is added between the classes 'student' and 'teacher', the identity index built before will be automatically extended to include instances of 'course' class. From the ROT only entries corresponding to the class 'teacher' are adjusted to show the course each 'teacher' instance is offerring instead of showing the students taking his course and each of the new entries that correspond to the 'course' class will refer to the students attending the course.

6. Conclusions

ODS supports the basic object-oriented concepts of object identity, class structure, inheritance, complex objects, encapsulation and schema modification, in addition to composite objects which is considered in [11] among the non-basic constructs. A schema modification methodology for ODS is described. The schema evolution functions are efficiently effected due to the flexibility that ODS gained from its particular storage and indexing models. The performance of the query processor is improved by physically dropping all references to a deleted chunk according to the object deletion rule, thus eliminating many instantiations otherwise required to recognize the absence of referenced chunks. Finally, because all the component chunks of a composite object are treated as a single chunk, we are currently addressing the problem of establishing a methodology for handling the rules and invariants for schema changes of such objects.

References

[1] Al-Hajj R., E. Arkun, A Model for Storage Management in Object-Oriented Database Management Systems, Proceeding of the Fifth International Symposium on Computer and Information Sciences, October 1990, Cappadocia , Turkey.

[2] Al-Hajj R., E. Arkun, A Model for Indexing in Object-Oriented Database Management Systems, Proceeding of the Fifth International Symposium on Computer and Information Sciences, October 1990, Cappadocia , Turkey.

[3] Atkinson M., et al., The Object-Oriented Database System Manifesto, Proceeding of the International Conference on Deductive Object-Oriented Databases, Kyoto, Japan, December, 1989.

[4] Banerjee J., et al., Data Model Issues for Object-Oriented Applications, ACM Transactions on Office Information Systems, vol.5, no.1, Jan.1987, pp.3-26.

[5] Banerjee J., et al., Schema Evolution in Object-Oriented Persistent Databases, Proc. of the 6th Advanced Database Symposium, Information Processing Society of Japan's Special Interest Group on Database Systems, Tokyo, Japan , August 1986, pp.23-31.

[6] Banerjee J., et al., Semantics and Implementation of Schema Evolution in Object-Oriented Databases, Proc. of ACM/SIGMOD International Conference on Management of Data, San Francisco, California, May 1987.

[7] Deux O., et al., The Story of O2, IEEE Transactions on Knowledge and Data Engineering, vol.2, no.1, March 1990, pp.91-108.

[8] Kesim N., Object Memory for an Object-Oriented Database Management System, M.S. Thesis, Bilkent University, Ankara, July 1988.

[9] Khoshafian S.N., and G.P. Copeland, Object Identity, Proceeding of ACM International Conference on Object-Oriented Programming Systems, Languages and Applicatoins, Sept. 1986.

[10] Kim W., H. Chou, and, J. Banerjee, Operations and Implementations of Complex Objects, IEEE Transactions on Software Engineering, Vol. 14, No. 7, July 1988.

[11] Kim W, Object-Oriented Databases: Definition and Research Directions, IEEE Transactions on Knowledge and Data Engineering, Vol.2, No.3, September 1990, pp.327-341.

[12] Kim W., et al., Indexing Techniques for Object-Oriented Databases, Object-Oriented Concepts, Applications, and Databases, W. Kim and F. Lochovsky, Eds., Addison-Wesley, 1989.

[13] Maier D., and J. Stein, Indexing in an Object-Oriented DBMS, Proc. of the Workshop on Object-Oriented Database Systems, September 1986.

[14] Maier D., A. Otis, and A. Purdy, Object-Oriented Database Development at Servio Logic, Database Engineering, IEEE, vol.8, no.4, December 1985.

[15] Özelçi M.S., Message Passing in an Object-Oriented Database Management System, M.S. Thesis, Bilkent University, Ankara, July 1988.

[16] Penney D.J., J. Stein, Class Modification in the GemStone Object-Oriented Database Management Systems, Proc. of Second International Conference on Object-Oriented Programming Systems, Languages and Applications, Orlando, FL, Oct. 1987.

[17] Skarra A.H., and S.B. Zdonik, The Management of Changing Types in an Object-Oriented Database, Proc. of ACM International Conference on Object-Oriented Programming Systems, Languages and Applications, Portland, OR, September 1986, pp.483-495.

[18] Skarra A.H., and S.B. Zdonik, Type Evolution in an Object-Oriented Database, Research Directions in Object-Oriented Programming, ed. B. Shiver, and P. Wenger, MIT Press Series in computer Systems, 1987, pp.393-415.

[19] Skarra A., et al., An Object Server for an Object-Oriented Database, Proceeding of ACM/IEEE International Workshop on Object-Oriented Database Systems, 1986.

[20] Stefik M., and D.G. Bobrow, Object-Oriented Programming: Themes and Variations, AI Magazine, January 1986, pp.40-62.

[21] Türkmen S., C. Yengül, E. Arkun, An Object-Oriented Database System Prototype, Proceeding of the Fourth International Symposium on Computer and Information Sciences, October 1989, Cesme, Turkey.

EDICION GRAFICA PARA INVIDENTES

Alonso Amo, F.; Montes, C.; Pazos, J.
Facultad de Informática - Universidad Politécnica de Madrid
Campus de Montegancedo
28660-Boadilla del Monte
Madrid

ABSTRACT

El procesamiento de los fondos bibliográficos para su utilización por parte de personas invidentes es, en la actualidad una tarea compleja y, en muchos casos, imposible. Para que este colectivo tenga acceso a libros de texto, estos han de ser convertidos previamente al alfabeto Braille, bien mecanografiándolos directamente o mediante un OCR, con la ayuda de un operador especializado. Es prácticamente imposible transcribir de forma automática a Braille texto matemático y científico, e incluso una persona altamente cualificada se ve obligada a realizar continuas consultas a los manuales, lo cual conlleva en la mayoría de los casos resultados con discrepancias semánticas que no pueden ser interpretadas por los invidentes.
En este artículo se describe un sistema de edición para invidentes (EDIE), que es capaz de convertir automáticamente a Braille tanto texto común como símbolos científicos y matemáticos, que aparecen en los fondos bibliográficos de las editoriales.

PALABRAS CLAVE: Ingeniería de Software, Traductores, Autómatas, Invidencia, Edición

1. Introducción

En la actualidad, hay un gran desarrollo en investigación y diseño de productos comerciales dirigidos al gran público para aumentar su calidad de vida, pero pocos esfuerzos se han orientado a la investigación de nuevas tecnologías que permitan a este público minoritario aumentar su calidad de vida e inserción laboral.

La situación descrita es un hecho reconocido por la comunidad científica, hasta el punto de que la propia Comisión Científica de las Comunidades Europeas ha lanzado recientemente el programa de investigación TIDE (Tecnología de la Comunicación, Control e Información para personas con discapacidades y ancianos) [12] encaminado al ámbito de la tecnología de la rehabilitación. Sus objetivos fundamentales consisten en el fomento de las ayudas técnicas, y la adaptación de la tecnología existente, para las personas discapacitadas y ancianos, con lo que coincide con la meta perseguida en el desarrollo de EDIE.

2 Características generales del editor

Se pretende definir un Sistema Editor capaz de dar acceso al invidente a documentos textuales y con simbología científica, sin perder su contenido semántico, a través de las herramientas informáticas de adaptación a invidentes. En concreto el sistema deberá:

- Analizar el texto de salida del proceso de edición producido por el paquete de edición y maquetación industrial de una editorial y, en base a los comandos reconocidos en el análisis, efectuar la ejecución apropiada.

- Traducir el texto de edición obtenido en la etapa anterior. Esta traducción deberá ser legible para personas videntes que utilicen el equipo, y comprensible para un invidente en la línea Braille, sintetizador de voz e impresora Braille.

- Formatear el texto en formato estándar para su adaptación a una futura edición por terminal, y a la impresora Braille.

- Producir un sistema de edición que permita el acceso al texto a través de la línea Braille y el sintetizador de voz.

3 El prototipo EDIE

Como resulta evidente, el Sistema Editor debe incorporar en su etapa de producción diferentes sistemas de edición y maquetación utilizados en editoriales y centros de documentación. Por ello, y de acuerdo con el ciclo de vida de la metodología IDEAL [10] [8], resulta necesario definir un sistema prototipo cuya implementación, haga viable el Sistema Editor.

EDIE es el sistema prototipo de edición en Braille, diseñado específicamente para tratar la información producida en soporte magnético por el paquete de edición y maquetación PENTA, y utilizando como adaptaciones el sintetizador de voz VERT PLUS o CIBER 232, la línea Braille BRAILLEX-IB 80, y la impresora Braille THIEL.

Se escoge el sistema PENTA por ser uno de los más potentes y el más utilizado por las grandes editoriales. En líneas generales, su funcionamiento es el siguiente: El texto se introduce y manipula utilizando terminales Penta Vue y Penta Saturn, el cual se compone y pagina a través del software Penta Quick y Penta Page. La salida final se envía al Mergenthaler Linitron, que produce la edición definitiva para su posterior impresión. Todo el hardware y software mencionado configuran el sistema Penta [9].

El sistema Penta presenta, asimismo, una complejidad y una completitud suficiente para mostrar un amplio abanico de situaciones comunes en el proceso de edición, por lo que sirve de guía para posteriores estudios de otros paquetes de edición.

La editorial seleccionada, y que utiliza este paquete, ha sido Santillana S.A., porque ofrece una amplia gama de publicaciones en todos los niveles de la enseñanza lo que posibilita al invidente el acceso inmediato a un gran volumen de fondos bibliográficos de carácter educativo.

Las adaptaciones Braille seleccionadas son las más utilizadas actualmente por el personal invidente en el entorno de un sistema personal de procesamiento, no obstante la sustitución por otras no planteará grandes cambios en el sistema prototipo.

4. Especificación del Sistema EDIE

Para la definición y posterior implementación del sistema prototipo EDIE, se fijaron los siguientes criterios de éxito:

a) Textos básicos, compuestos por caracteres alfanuméricos y de puntuación. En primera fase, debe reconocer la estructura y significado de todos los caracteres del texto fuente. En segunda fase, debe traducir el texto de entrada a caracteres Braille de salida o a órdenes de formato, dependiendo del significado de la entrada.

b) Textos científicos, compuestos por texto básico y simbología científica. El sistema debe realizar las fases mencionadas en el punto anterior con este tipo de texto.

Los anteriores criterios dieron lugar a las siguientes especificaciones:

- Reconocimiento y procesamiento de texto básico generado por Penta.
- Traducción a código ASCII-Braille.
- Ejecución de órdenes de formateo Penta en texto básico de edición.
- Reconocimiento y procesamiento de texto científico Penta.
- Traducción a código ASCII-Braille científico.
- Ejecución de órdenes de formateo Penta en texto científico de edición.

Estas especificaciones configuran el diagrama de flujo de datos de la fig. 1.

5. Estructuras de proceso y de datos del Sistema EDIE

Realmente la información inicial contiene dos tipos básicos de elementos de información a considerar: el texto, que se obtiene como producto final del proceso de edición, y los comandos de control, que permiten al paquete de edición, mediante un reconocimiento de los mismos, realizar el diseño y la maquetación. El sistema ha de permitir, pues, extraer el texto explícito e implícito mediante una interpretación adecuada de los comandos de control, de manera que se mantenga intacta su semántica.

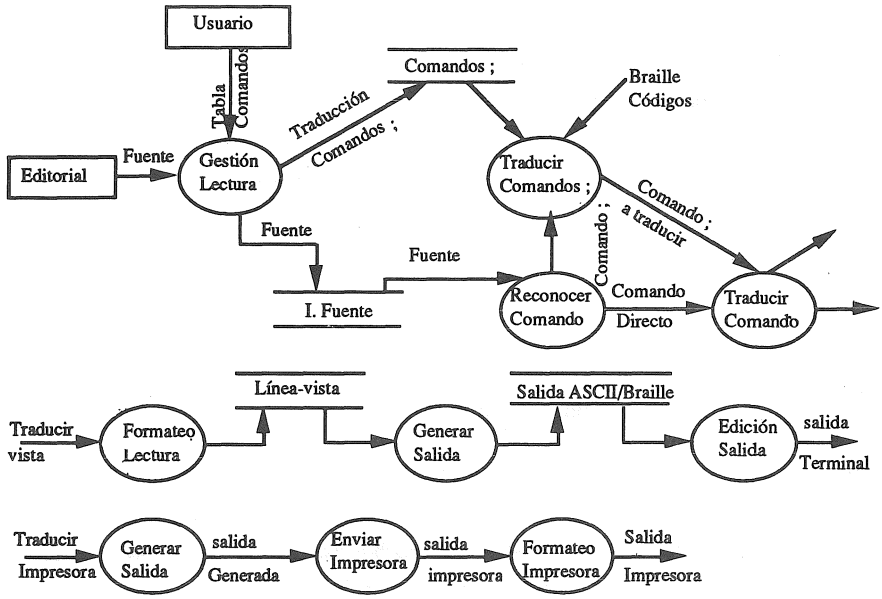


Figura 1. Flujo de datos del Sistema EDIE

Por ello se definieron unas estructuras específicas de información en base a la estructura de estos comandos de control con el fin de realizar un reconocimiento de las acciones que toman sobre el texto final, y así poder efectuar las acciones adecuadas para el mantenimiento de la integridad en la salida.

Este reconocimiento se realiza mediante **un autómata finito** de reconocimiento de cada comando, y a unas tablas **hash** abiertas por los comandos ";".

El problema a tratar es claramente de reconocimiento y traducción por medio de un lenguaje regular, cuya extensión y sintaxis es conocida. Este lenguaje regular es representable por una gramática de tipo 3 de Chomsky [3] [5] [2] en la que las producciones quedan restringidas a:

$$A \longrightarrow \alpha$$
$$A \longrightarrow a\beta$$

donde el lado derecho consta de un símbolo terminal, o de un símbolo terminal seguido por un símbolo no terminal; y se reconoce por una estructura de autómata finito.

En el caso concreto que nos ocupa, la quíntupla (E,Q,f,p_0, F) que define el autómata finito tiene el siguiente significado:

E Alfabeto de entrada compuesto por texto y comandos.

Q Conjunto, no vacío de estados, que se agrupan en dos niveles

- Nivel Principal: comprende 8 estados según el tipo de comando (fig. 2)

- Nivel Comando: cada uno de los estados de un comando (fig. 3)

f Función de transición de Q x E (fig. 3)

p_0 Estado inicial del autómata con valor cero.

F Conjunto de estados finales (fig. 3)

Tipo	Comando
1	'='
2	';'
3	'['
4	'-' y '^'
5	'I'

Figura 2.Tipos de comandos

El autómata finito determinista queda representado por el diagrama de transición de la figura 3, grafo cuyos nodos son los elementos Q y cuyas ramas dirigidas son función de f. Su estructura es especialmente útil en este caso en que se tienen muchos estados [7], y se conoce, en general, su sintaxis y semántica. No obstante, los comandos de tipo 2 (de ;) presentan la peculiaridad de que se conoce su sintaxis, pero es desconocida para el autómata su número (que en principio es ilimitado), y su semántica. Ello es debido, a que vienen definidos por el usuario, de acuerdo con la utilización que se prevea, dando lugar al tratamiento de un lenguaje variable.

Para el reconocimiento de este tipo de lenguajes las técnicas a aplicar son las redes ATN y los diccionarios. En este caso se implementó un diccionario conteniendo los comandos y su traducción, en una estructura de tabla hash abierta [1] [6] [11], y utilizando el comando como clave. La función hash será de la forma:

hash (comando ;) = (comando ;) - ' ; ^'

asignando a cada comando el código ASCII del primer carácter posterior a ' ; ' menos el valor ASCII ' ^ ', primer valor que se encuentra después de punto y coma. La tabla se ubicará en memoria principal para conseguir una velocidad razonable de traducción, y la carga de sus valores se efectúa mediante estructuras dinámicas lo que permite el acceso inmediato a estos comandos.

Las principales ventajas de este diseño están en que permiten una inserción rápida en tiempo de ejecución de nuevos comandos, efectúa una gestión dinámica de memoria reservando espacio para un nuevo comando sólo en el momento de su aparición, y proporciona un buen tiempo de respuesta en la ejecución del comando.

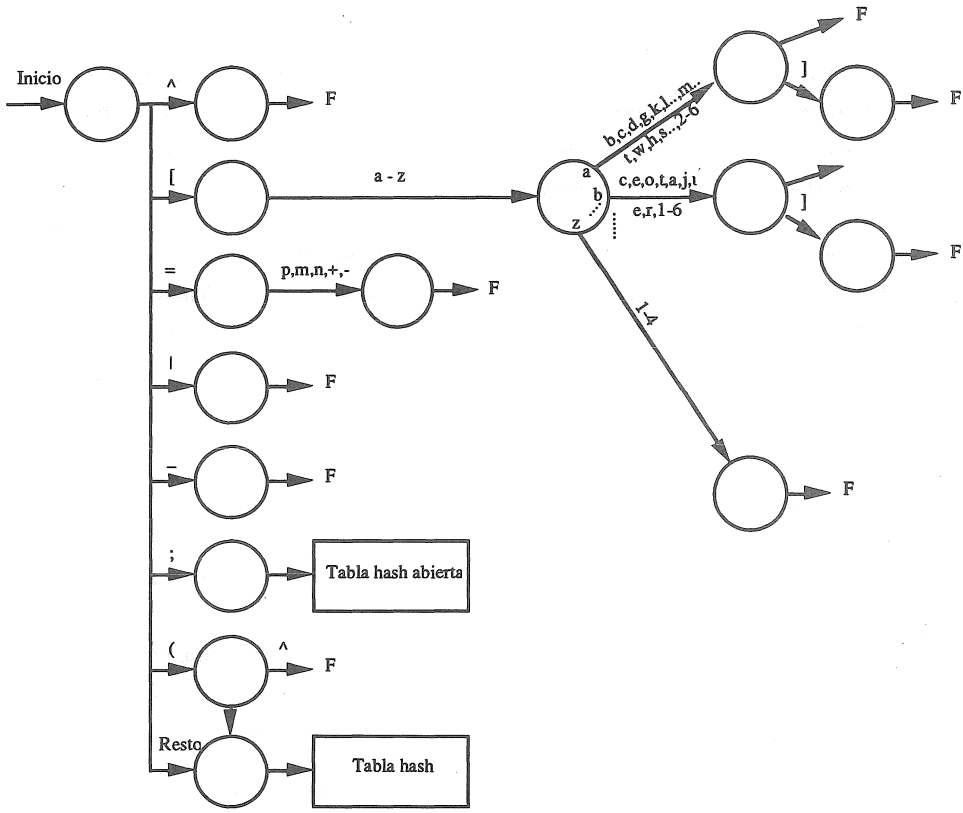


Figura 3. Autómata finito para reconocimiento de comandos

Otra estructura importante, en el diseño del sistema, es la traducción a lenguaje Braille de la correspondiente codificación ASCII. En este punto interesa destacar los diferentes sistemas Braille existentes, por lo que se ha utilizado un diccionario de traducciones, implementado mediante una tabla hash de símbolos [1] [6], siendo la función hash:

hash (carácter) = carácter - ' '

donde la clave la constituye un carácter, y la información la codificación Braille correspondiente. Se está ante un conjunto finito y conocido de entradas al diccionario, que permiten la modificación de las traducciones en función de los diferentes sistemas Braille que se utilicen. En el caso que nos ocupa son dos, uno para la línea Braille y otro para la impresora Braille.

6. Edición en línea Braille y sintetizador de voz

El análisis de estas adaptaciones conduce a las siguientes conclusiones:

- La línea Braille es un dispositivo, fundamentalmente hardware, que efectúa una captura de los caracteres visualizados en la línea de pantalla donde se sitúa el cursor. Por ello, no es posible representar en línea Braille nada que no se represente en la pantalla, al menos sin perder alguna de las dos representaciones.

- El sintetizador de voz es un dispositivo, principalmente software, que captura de pantalla los caracteres de la línea en la que se encuentra el cursor. Lee por sílabas, y en el caso de detectar un carácter desconocido deletrea todos los caracteres. Dispone de un juego reducido de caracteres, inferior al conjunto ascii, y por supuesto, no reconoce los caracteres no-ascii.

- En el teclado del sistema se dispone de todos los caracteres ascii, pero no incluye la mayoría de los símbolos propios de las matemáticas, lógica, química, etc.

Teniendo en cuenta, que se trata de editar un texto compuesto por caracteres ascii y notación científica, y considerando las diferentes limitaciones presentadas por cada adaptación para invidentes, se ha optado como solución la de transcribir con la palabra completa el nombre del símbolo (ej. $\exists$ por 'existe'). De esta forma se llega a una notación legible para las tres representaciones.

Para el manejo del texto de salida se pensó, inicialmente, utilizar un editor comercial, pero el hecho de que el sintetizador de voz requiera que el cursor esté posicionado en la línea que se desea leer, obliga a desechar esta opción, y a tener que diseñar un editor propio, sólo de lectura, que permitan la edición del texto convertido, y el libre movimiento del cursor por el principio de todas las líneas del texto, resolviendo así la lectura del sintetizador.

7. Edición en impresora Braille

La edición con impresora Braille (THIEL) se trata independientemente por ser un elemento básicamente de producción Braille, más que de tratamiento interactivo.

Esta impresora presenta dos modos de funcionamiento [13]:

a) Modo ascii.- Cuenta con un generador de código que traduce los caracteres ascii a código Braille.

b) Modo transparente.- Los datos que recibe son interpretados de derecha a izquierda como caracteres Braille, asignado el bit 0 del carácter al punto 1 de la matriz Braille, el bit 1 al punto 2,... y el bit 7 al punto 8.

La impresora permite pasar de un modo a otro mediante secuencias de escape.

Con la impresora funcionando en modo ascii se diseñará un sistema de conversión Braille de todos los caracteres, incluidos los de notación científica, a código Braille, conforme con el Código Matemático Unificado [4], produciendo este dispositivo una salida Braille que representa fielmente los textos normales y científicos que se traduzcan.

Este sistema de edición en impresora incorporará en este prototipo un formateador estándar Braille para líneas, saltos de líneas, páginas, nº de página, encabezamiento,... de acuerdo con la normativa Braille.

8. Resultados y conclusiones

El sistema prototipo EDIE permite al invidente el acceso a la documentación bibliográfica textual y científica, sobre soporte magnético, producida por el sistema PENTA de edición y maquetación industrial. Ello posibilita la utilización de los fondos editoriales de Santillana y SM, por citar dos importantes empresas del sector que utilizan el sistema PENTA, así como los de otras editoriales que trabajan con este sistema.

Finalizada esta etapa, el Sistema Editor tratará bibliografía producida en entornos Macintosh que está empezando a introducirse en el mundo de las editoriales.

La implementación del sistema se ha realizado en C++, utilizándose la estructura orientada a objetos a nivel de módulo.

En resumen, el desarrollo del Sistema Editor que se describe en este artículo ha planteado, en particular, un análisis específico de los siguientes elementos y estructuras:

- Sistema Penta. Estudio general del Sistema, y dominio de los comandos de control y traducibles (o comandos ;).

- Autómata finito determinista, para el reconocimiento y posterior tratamiento de los comandos de control.

- Tabla hash de listas encadenadas, para el reconocimiento del lenguaje variable de los comandos;

- Diccionario de traducciones, implementado mediante una tabla hash de símbolos, para la traducción a lenguaje Braille de la correspondiente codificación ASCII.

- Línea Braille.- Estudio de este dispositivo, de su código Braille, y su intersección con la pantalla del Ordenador.

- Sintetizador de voz.- Estudio de este dispositivo, e intersección con la pantalla del ordenador.

- Impresora Braille.- Estudio de los modos de funcionamiento ASCII y transparente, y del Código Matemático Braille que se utiliza para la notación científica.

EDIE ha deparado unos resultados muy satisfactorios, hasta el punto de que está siendo utilizado en el Centro de Producción Bibliográfica de la Organización Nacional de Ciegos.

El grado de fidelidad en la transcripción del texto procedente del sistema PENTA alcanza el 100%. Esto quiere decir que si el operario encargado de la maquetación utilizara PENTA perfecta y precisamente, el problema de la conversión estaría resuelto. Ahora bien, en multitud de ocasiones los maquetadores hacen "trampas" al sistema con el fin de simplificar su trabajo (ej. utilizar un código de tabla para recuadrar una frase o no utilizar subíndices sino cambios de tamaño del fuente), lo que conlleva la aparición de una serie de ambigüedades e imprecisiones en el resultado final.

Para resolver este problema, se está desarrollando en la actualidad un filtro-maquetador inteligente, capaz de resolver situaciones ambiguas dependientemente del contexto, y de producir una maquetación del texto adecuada para los invidentes.

En cuanto al tratamiento de los gráficos y figuras que aparecen en el texto, su tratamiento es dependiente de su complejidad. Los gráficos y figuras simples actualmente se reconocerán e interpretarán, generando automáticamente la salida de la gráfica y su descripción. Los gráficos complejos serán referenciados para su incorporación manual, ya que en la actualidad no existe posibilidad de desarrollar un reconocedor universal de imágenes.

9. Bibliografía

[1] Alonso-Amo, F.; Morales Lozano, A.: *Técnicas de Programación - Programación Estructurada, PASCAL, Estructuras de Datos*. Paraninfo. 1987.

[2] Chomsky, A.N.: *On Certain Formal Properties of Grammars*. Information Control, Vol. 2, June 1959.

[3] Crespo: *Informática Teórica*. Vol. 1. Facultad de Informática. UPM. 1982.

[4] Della Barca, J.J.; Ortiz, M.A.; Rodrigo, F.; Roig, C.: *Código Matemático Unificado para la Lengua Castellana*. Reunión de imprentas Braille de habla hispana. Montevideo, 1987.

[5] Hopgood, F.R.A.: *Compiling Techniques*. Macdonald. London and American Elsevier, Inc. 1970.

[6] Horowitz, E.; Sahni, J.: *Fundamentals of Date Structures in Pascal*. Computer Science Press. 1984.

[7] Kain, R.Y.: *Automate Theory: Machines and Languages*. McGraw-Hill. 1972.

[8] Maté, J.L.; Pazos, J.: *Ingeniería del Conocimiento. Diseño y Construcción de Sistemas Expertos*. Facultad de Informática, Madrid, 1988.

[9] Pazos, J.: *Metodología IDEAL para el Diseño e Implementación de Sistemas Expertos*. Actas de las Jornadas de Rank Xerox sobre Sistemas Expertos, 1986.

[10] PENTA: *Command Reference Manual*. Penta Systems International INC, 1985.

[11] Sanchis, F.J.; Galán, C.: *Compiladores. Teoría y Construcción*. Paraninfo, S.A. 1981.

[12] TIDE: *Documentación del Proyecto TIDE de la Comunidad Europea :Information Package*. 1990.

[13] THIEL: *Manual de utilización e instalación de la Impresora Braille THIEL*. Thiel Gmble Co.,1984.

Compilación Optimizante para el Lenguaje Occam sobre un Sistema Time-Sharing

Guillermo A. Alvarez * Daniel O. Bandinelli *

Marcelo O. Fernández * Gustavo A. Rosini *

Resumen: Occam es un lenguaje de programación basado en un modelo de procesos secuenciales que se ejecutan concurrentemente, comunicándose por medio de pasaje sincrónico de mensajes. Se detallan los aspectos del desarrollo de un compilador Occam–C sobre una máquina uniprocesador. El sistema operativo UNIX es usado como plataforma de implementación. Se incluyen explicaciones detalladas de los chequeos realizados, de los procedimientos de optimización, y de las técnicas de compilación desarrolladas. Se define la traducción de un lenguaje diseñado para arquitecturas multiprocesador sobre sistemas monoprocesador time-sharing.
Palabras clave: Compiladores, Generación Optimizante de Código, Lenguajes de Programación, Programación Concurrente, Sistemas Operativos.

1 Introducción

Durante las últimas dos décadas, diferentes paradigmas de programación paralela han sido propuestos. Entre ellos se encuentran los paradigmas basados en pasaje de mensajes como Occam [7]. Esta familia de lenguajes tiene su sustento teórico en CSP [5], y admite como soporte a arquitecturas multiprocesador con memoria distribuida y pasaje de mensajes.

Este trabajo describe el desarrollo de un compilador para el lenguaje concurrente Occam que genera código C sobre el sistema operativo UNIX System V. Si bien el lenguaje Occam fue concebido originalmente para ser ejecutado en un array de procesadores homogéneos (Transputers), en nuestro caso se simula la ejecución paralela de procesos secuenciales utilizando las facilidades provistas por UNIX para programación concurrente y comunicación entre procesos.

El compilador acepta un subconjunto del lenguaje Occam estándar con la adición de extensiones destinadas a facilitar la interacción con el usuario. Dichas extensiones permiten entrada/salida respetando el modelo conceptual del lenguaje.

La compilación de un programa consiste en dos etapas que se realizan secuencialmente. La primera (*front-end*) acepta como entrada el texto de un programa Occam sintácticamente correcto que cumple las restricciones inherentes al lenguaje, y lo transforma en código para una máquina abstracta. La segunda (*back-end*) toma la salida de la primera y genera código para una máquina real en términos del código para la máquina abstracta. Cada una de estas etapas se efectúa en una pasada sobre el código, y por lo tanto el proceso total de compilación insume solamente dos pasadas.

El trabajo está organizado como sigue. Las Secciones 2 y 3 detallan aspectos del front-end y del back-end, respectivamente. En la Sección 4 se explican las optimizaciones implementadas; por último, en la Sección 5 se presentan las conclusiones.

2 Chequeos y Generación de Código Intermedio

En esta sección se detallarán los diversos chequeos realizados sobre el programa Occam de entrada. Para realizar dichos chequeos, se debe recoger durante la fase de análisis del texto del

*Escuela Superior Latino Americana de Informática (ESLAI) – CC 3193 – (1000) Buenos Aires – Argentina; e-mail alvarezg@buevm2.vnet.ibm.com, danielb@dcfcen.edu.ar

programa la siguiente información relativa a los identificadores: nombre, tipo (variable, canal o constante), valor (para las constantes), alcance, e información de posición. La información de posición denota qué tipo de accesos al identificador se hacen y en qué lugares (procesos) del texto fuente se los hace. Es usada primordialmente para garantizar que la comunicación entre procesos se realize utilizando canales y no variables compartidas. Además sirve para chequear la correcta utilización de los canales.

Por intermedio de los chequeos realizados hemos podido determinar tanto errores fatales como warnings que ayudan al programador a realizar un código más confiable. Los chequeos realizados son detallados a continuación.

Utilización de variables compartidas: No permitir que dos procesos que pueden ejecutarse en paralelo se comuniquen a través de variables compartidas.

Utilización de canales: Dados dos procesos que pueden ejecutarse en paralelo, verificar que sólo uno envía y sólo uno recibe mensajes por un canal.

Constantes, variables y canales no utilizados: Se detectan los identificadores declarados pero no utilizados en el texto Occam, notificándose con el warning correspondiente.

Variable que es leída y no está inicializada: También se avisa (a través de un warning) de esta situación para prevenir al programador de un comportamiento posiblemente no determinístico del programa.

Los identificadores utilizados deben estar declarados una sola vez: Se chequea que todo identificador esté declarado en el alcance en que se lo utiliza. Además no permitimos declaraciones de un identificador cuando ya haya sido declarado en el mismo alcance con un tipo distinto. La violación de cualquiera de estos dos requisitos resulta en un error.

Uso de un identificador de acuerdo a su tipo: Se verifica que los identificadores se usen consistentemente con sus declaraciones, por ejemplo que en una escritura por canal no se pretenda escribir el dato sobre una variable entera.

La expresión que aparece en un proceso `WAIT` debe ser una comparación de tiempos: Sólo se permiten procesos `WAIT` de la forma `WAIT NOW AFTER exp` donde `exp` no contiene ocurrencias de `NOW`.

Un solo tipo de operador asociativo debe aparecer en una expresión no parentizada: No se permite que una secuencia de expresiones esté operada con operadores asociativos distintos, a menos que esté máximamente parentizada.

Para escribir el front-end se utilizaron el generador de analizadores lexicográficos `lex` y el generador de parsers `yacc` provistos por UNIX. Estos brindan una forma clara y cercana al modelo teórico de especificar el lenguaje que se desea reconocer usando esquemas de traducción [1]. Además permiten modificar fácilmente la gramática, lo cual no ocurriría si tanto el parser como el analizador lexicográfico estuvieran implementados directamente en C. El hecho de utilizarlos para el desarrollo del compilador favorece su portabilidad a otros sistemas UNIX.

El código intermedio generado como consecuencia de la primera etapa de la compilación tiene la forma de un árbol sintáctico, que refleja la estructura del programa fuente después de que se le realizan las optimizaciones detalladas en la Sección 4.

3 Generación de Código

La presente sección contiene aspectos relativos a las prestaciones que brinda el soporte de tiempo de ejecución implementado como complemento del compilador, y a la generación de código C a partir de los principales constructores Occam.

3.1 Soporte de Tiempo de Ejecución

La traducción de Occam a C involucra la utilización de facilidades de sincronización y comunicación entre procesos, por lo cual el soporte de tiempo de ejecución permite crear y liberar

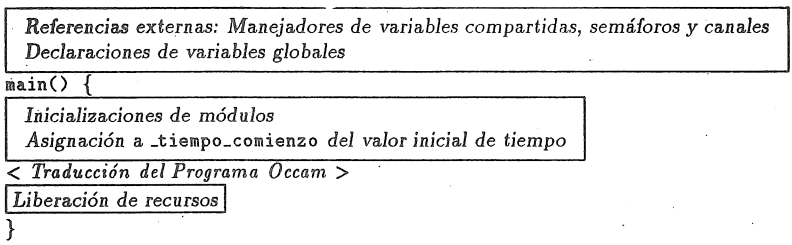


Figure 1: Formato Típico del Código C Generado.

dinámicamente dichos recursos en forma eficiente. Esto se logra a través de manejadores que se encargan de administrar el uso de canales de comunicación, variables en memoria compartida y semáforos.

El manejador de canales se construyó usando las facilidades provistas por los manejadores de semáforos y de memoria compartida. La comunicación entre procesos es unidireccional, es decir hay un proceso emisor del mensaje (en Occam este mensaje es un entero) y el otro proceso es el receptor, no pudiéndose intercambiar ambos papeles. Además la comunicación es sincrónica, esto implica que se lleva a cabo cuando ambos procesos (el emisor y el receptor) quieren comunicarse. Si sólo uno de los procesos está listo para la comunicación debe esperar a que el otro también lo esté.

De los identificadores Occam, se marcan como residentes en memoria compartida sólo a aquellos que probablemente se usen para comunicar resultados de un proceso hijo a su padre. Esto ocurre cuando un componente de un constructor PAR modifica una variable que es leída posteriormente por un proceso que se ejecuta después de que el PAR termina. Dado que UNIX no permite crear variables compartidas individuales (porque se debe crear un segmento de memoria compartida cuyo tamaño debe ser mayor que un límite fijado por el sistema operativo [3]), se debe implementar un manejador intermedio que permita efectuar pedidos de memoria con mayor frecuencia y granularidad más fina. Como las variables compartidas van a ser usadas por varios procesos simultáneamente, las estructuras utilizadas por el manejador deben estar en memoria compartida para poder ser accedidas por todos los procesos.

Para manejar semáforos se han definido operaciones según la notación usual (definida en [4]). Se debe tener en cuenta que UNIX [9] no maneja semáforos individuales sino arreglos de los mismos. Este problema es similar al visto anteriormente para administrar las variables compartidas, y se soluciona implementando un manejador que sigue los mismos principios.

Las variables compartidas y los canales del programa Occam determinan la generación de dos clases de código: declaraciones para los identificadores C asociados a ellos, y código ejecutable C que invoca a los respectivos manejadores, creando y liberando los recursos cuando los correspondientes identificadores entran y salen de alcance.

3.2 Estructura del Generador de Código

Dado un árbol que representa el código intermedio del programa Occam, se lo recorre recursivamente generando el correspondiente código C. Para ello se genera un encabezamiento que produce la inclusión de los manejadores de semáforos, canales y variables compartidas, y además declara algunas variables globales del soporte de tiempo de ejecución. Luego se genera la declaración de la función `main()` donde se inicializan los manejadores y las variables del soporte, y a continuación el código correspondiente al árbol; se termina con el cierre de los módulos utilizados.

El formato de un programa C típico, generado como salida del back-end, es mostrado en la Figura 1.

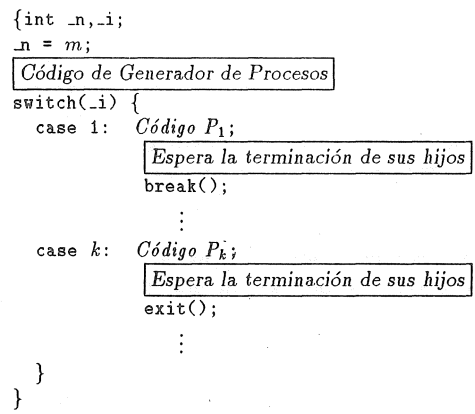


Figure 2: Traducción del constructor PAR.

Declaraciones: Al procesar el código intermedio correspondiente a un bloque de declaraciones Occam, se genera un bloque C donde las variables Occam se declaran como variables enteras si no son compartidas. Para los canales, generamos una declaración de tipo `_chan`.

Después de generar las declaraciones del bloque C y antes de comenzar con el código, se generan los `create _vc()` y los `create _chan()` correspondientes, para que los manejadores creen los recursos en tiempo de ejecución. Análogamente, al terminar el procesamiento del bloque, generamos los `remove _vc()` y `remove _chan()` correspondientes antes de cerrar el bloque C.

No es necesario generar declaraciones para las constantes Occam en esta etapa, ya que al construir el árbol las apariciones de las constantes se reemplazaron por el valor correspondiente.

Cabe notar que se generan únicamente las declaraciones correspondientes a aquellas variables y canales que realmente fueron usados, y no para los que fueron declarados pero no utilizados, lográndose de esta manera el consecuente ahorro de espacio de almacenamiento y de costo de creación al ejecutar.

Constructor PAR: La traducción del constructor PAR consiste en generar código que crea dinámicamente procesos UNIX, cada uno de los cuales ejecuta uno de los procesos componentes del constructor. La creación de procesos se hace en forma de árbol binario completo por niveles para mejorar la eficiencia, como se explicará en la Sección 4.

Por lo tanto, una construcción Occam de la forma PAR P_1 ... P_m se traducirá, si $m > 1$ (si no, se genera código directamente para el único proceso), en el código C de la Figura 2.

La variable `_n` indica al generador la cantidad de procesos a crear; a la salida, `_i` contiene el número de orden de cada proceso generado, por lo cual puede elegir la rama correcta del `switch()` y ejecutar su código. Cada proceso genera a lo sumo dos hijos (según su posición en el árbol), ejecuta su rama del PAR y luego espera que sus hijos terminen. El proceso padre ejecuta la primera componente del constructor. Dentro del código del generador se tienen en cuenta posibles errores de ejecución al excederse la cantidad máxima de procesos permitida por el sistema.

Constructor ALT: Si alguna de las guardas del ALT se revela verdadera en tiempo de compilación, entonces se genera solamente el código del proceso guardado correspondiente, como una implementación del no-determinismo.

La política que se adoptará al evaluar las guardas será considerarlas como una generalización del concepto de expresiones booleanas: una guarda (en el caso más complejo) será cierta cuando la expresión booleana que está a la izquierda del & sea cierta, y además la operación que está a la derecha pueda realizarse. En este contexto, esta expresión se evaluará siguiendo la semántica estándar de Occam, de izquierda a derecha y con corto circuito. Por lo tanto, al ejecutarse

el código generado a partir de un ALT, se evaluarán primero todas las expresiones que figuren en las partes izquierdas de guardas, y se descartarán definitivamente las que den FALSE. Esta decisión posibilita una implementación en la que ninguno de los casos posibles provoca espera activa, como se verá más adelante, ya que aún en el caso de expresiones que involucren a la función primitiva NOW su valor de verdad se evaluará una sola vez.

Tras este paso inicial de evaluación, pueden existir guardas de la forma `exp & SKIP` cuya expresión haya evaluado a TRUE, y que por lo tanto sean ciertas. Es por esto que se genera código para verificar al ejecutar si esto ocurre, y en el caso de que ocurra se dice que el ALT se ha *trivializado*: se ejecutará el proceso guardado correspondiente.

Las guardas que pasaron la primera etapa (sus expresiones son verdaderas, o bien consisten en una entrada por canal o WAIT aislados) son de dos clases distintas, asumiendo que el ALT no se ha trivializado: entradas o procesos WAIT. De aquí en adelante sólo nos referiremos a estas guardas. Para implementar su evaluación, se genera en tiempo de corrida un proceso para cada una de estas guardas. La generación de procesos se hace en forma plana y no de árbol binario ya que si se mantuviera la estructura de creación de árbol binario, podría pasar que algunos procesos fueran descartados por ser falsas las expresiones de sus guardas, pero éstos no generarían a sus hijos en el orden arbóreo, que sí podrían tener guardas válidas. Dicha generación de procesos se hace sólo en los casos en que el ALT no se ha trivializado y no todas las guardas tienen expresiones falsas como componente.

Ninguno de estos procesos hace espera activa, ya que los encargados de evaluar WAITs calculan la cantidad de tiempo que tendrá que pasar y ejecutan un `sleep()` de esa duración; por otra parte, los encargados de evaluar entradas se duermen mientras no haya un dato disponible en el canal. Una implementación alternativa para el ALT hubiera sido evaluar continuamente las guardas hasta que alguna se cumpliera, desperdiciando tiempo de CPU en actividades inútiles. Fue necesario definir un mecanismo para que el proceso cuya guarda se cumpla primero avise al proceso padre (coordinador) que debe ejecutar su correspondiente proceso guardado. Para ello, el padre genera primero los procesos evaluadores de guardas y luego se queda suspendido en el semáforo `_alguien _termino` (que inicialmente vale 0). Cuando un proceso hijo puede cumplir su guarda, verifica primero si puede avisarle al padre, para garantizar que el padre aceptará a sólo un proceso; si puede, dejará su número en un espacio de memoria compartida, y luego ejecutará un `V(_alguien _termino)` despertando por lo tanto al padre. El padre, una vez enterado de que todos sus hijos instalaron sus manejadores (a través del semáforo `_activados`), espera hasta que uno de ellos triunfe; sólo entonces interrumpe enviando señales a los demás procesos hijos para que éstos terminen su ejecución. Esto se hace para no esperar a guardas demasiado lentas, y para reducir la cantidad de procesos en el sistema al mínimo. Un hijo que satisface su guarda ejecuta un `P(_primero _que _termina)` (inicialmente en 1), para que si dos guardas se cumplen al mismo tiempo sólo una pueda avisar al padre y la otra quede suspendida en dicho semáforo.

Los procesos que esperan entradas sobre el mismo canal necesitan que sólo uno de ellos retire un dato de ese canal; para lograr esto, se define un semáforo auxiliar por cada canal (`_sem` *_nomcanal*) inicialmente en 1, que sólo permite que una guarda ejecute la lectura sobre dicho canal.

Los procesos que esperan entradas sobre canales distintos deben mantener la semántica de Occam en el sentido de que sólo el que logra ejecutar su proceso guardado retira el dato del canal, dejando los demás canales sin cambio (en ellos no se retira el dato ni se desbloquea al proceso que lo envió), y además tampoco altera las variables que figuran como destino en las entradas no exitosas. Para solucionar el problema, se define un `receive _guard()` sobre canales, el cual no desbloquea al proceso que envió el dato hasta haberle avisado al padre que él es la guarda verdadera. Y para no alterar variables destino de entradas supuestamente no ejecutadas, se definen variables auxiliares sobre las cuales realmente se hace la entrada. Al comienzo del proceso guardado, se asignan estas variables a la verdadera variable destino. Por lo tanto, la

evaluación de guardas a cargo de procesos independientes no causa efectos laterales indeseables.

Cuando un proceso evaluador de guarda es interrumpido por su padre, pasa a ejecutar un manejador, que en el caso de los `WAIT` y de las entradas desde el `stdin` se limita a abortar la ejecución, pero en el caso de las entradas desde canales Occam se fija si estaba a la mitad de la lectura, restaurando en ese caso el estado del canal.

Por lo tanto, una construcción Occam de la forma `ALT` $\begin{matrix} G_1 & \dots & G_m \\ P_1 & & P_m \end{matrix}$ se traducirá en el código C de la Figura 3, donde *nomvar* es la variable sobre la cual se realiza la entrada.

Demás Constructores: Los constructores `SEQ`, `IF` y `WHILE` se traducen de la forma natural. En cuanto a las lecturas y escrituras de canales, se realizan a través de las primitivas sincrónicas del manejador de canales, salvo en el caso de `IN` y `OUT`. Para éstos se generan llamadas a las rutinas estándares de C para entrada/salida.

4 Optimizaciones

En esta sección se mencionan las optimizaciones implementadas como parte del compilador. Dichas optimizaciones se dividen básicamente entre las que se realizan en tiempo de compilación (específicamente dentro del front-end) y las que se realizan en tiempo de ejecución (consecuencia del código generado especialmente por el back-end).

Como optimizaciones implementadas en la primera etapa pueden destacarse las siguientes:

- Evaluación y propagación estática de constantes, teniendo en cuenta los aspectos de evaluación de izquierda a derecha (corto circuito) y de coerción de entero a booleano.
- Simplificación del código fuente, eliminando estructuras de control cuyo comportamiento es predecible estáticamente (por ejemplo, un `WHILE` cuya condición evalúa estáticamente a `FALSE`) y manejando los casos de construcciones vacías. Se manejan los casos en que la eliminación de partes inútiles de código convierte a su vez en inútiles a los constructores que las encierran. Esto se hace en base a la evaluación estática de constantes.
- Simplificación de estructuras anidadas con constructores del mismo tipo, para evitar el uso excesivo de recursos atribuibles a la implementación de dichos constructores.
- Minimización del uso de variables en memoria compartida, marcando como tales sólo a las que probablemente se usen para comunicar resultados de un proceso hijo a su padre. Esto ocurre cuando un componente de un constructor `PAR` modifica una variable que es leída por un proceso que se ejecuta después de que el `PAR` termina. Se descartó la alternativa de mantener todas las variables en memoria compartida porque implicaba un nivel adicional de indirección para cada acceso, más las llamadas al manejador de memoria compartida para reservar y liberar espacio.
- Minimización del costo por llamadas al manejador de memoria dinámica de C durante la compilación, implementando un nivel adicional que sigue una heurística orientada a no liberar memoria hasta que sea altamente probable que no se la necesite de nuevo. Esto es utilizado para hacer más eficiente el procesamiento de los bloques de declaraciones.
- Construcciones que extienden a Occam, mediante canales de comunicación con la entrada y la salida estándares de UNIX.
- Generación de código intermedio declarando identificadores sólo para aquellos que efectivamente se usan en el código Occam.
- Mensajes de error y warning destinados a facilitar el proceso de depuración de programas Occam y el uso racional de recursos, avisando de la eventual no terminación de programas, variables no utilizadas, variables no inicializadas, etc.

```
{int _i,_n,_proc_guard = 0,_alguna_exp_cierta = 0,_alt_simplificado = 0;
_semaforo _alguien_termino,_primero_que_termina,_activados;
_vc _termino;
Declaraciones de Otras Variables y Semáforos Auxiliares
create_vc(&_termino);
create_sem(&_alguien_termino,0);
create_sem(&_primero_que_termina,1);
Creación e Inicialización de Variables y Semáforos Auxiliares
_n = m ;
Evaluación de las Expresiones Booleanas
if (not _alt_simplificado || _alguna_exp_cierta){
   create_sem(&_activados,m);
   Generador Plano de Procesos
   switch(_i){
       case 1:Código G1
              exit();
                 ⋮
       case m+1:P(&_alguien_termino);
                _proc_guard = read_vc(_termino);
                wait_for_zero(&_activados);
                  Enviar Interrupción a los Hijos no Exitosos
                  Esperar Fin de Hijos
       }
   }
switch(_proc_guard) {
   case 1:Código P1
          break;
           ⋮
   case m:Código Pm
          break;
   }
remove_vc(_termino);
Liberación de Variables y Semáforos
}
```

(a)

```
signal(ABORT,_handler_wait);
P(&_activados);
Código de WAIT
P(&_primero_que_termina);
write_vc(_termino,_i);
V(&_alguien_termino);
```

(b)

```
signal(ABORT,_handler_entrada);
P(&_activados);
P(&_sem_nomcanal);
receive_guard(&_nomcanal,&_primero_que_termina,
  &_alguien_termina,__nomvar,_i);
```

(c)

Figure 3: Constructor ALT: (a) Código Principal; (b) Código de un Proceso Evaluador de Guardas WAIT; (c) Código de un Proceso Evaluador de Guardas con Entradas desde Canales.

en tanto que en la segunda etapa se destacan:

- Utilización de manejadores para disminuir la cantidad de llamadas al sistema operativo aumentando la eficiencia, y también para proveer servicios de la granularidad necesaria para la ejecución del código C generado.
- Generación paralela asincrónica de procesos en tiempo de corrida, utilizando un esquema recursivo que extrae el máximo paralelismo del sistema time-sharing. Básicamente, en vez de hacer que el proceso padre genere a todos sus hijos y espere a que terminen para seguir, cada proceso genera a 0, 1 o 2 hijos y luego realiza su parte del cómputo. Debido a que la creación de un proceso involucra entrada/salida de periféricos del sistema, esta estrategia mejora la eficiencia creando los procesos según una estructura de árbol binario que aprovecha el paralelismo entre cómputo y entrada/salida.
- Minimización de la cantidad de procesos generados dinámicamente, de dos formas distintas: haciendo que el proceso padre en un constructor `PAR` ejecute una parte del código en vez de esperar a que sus hijos terminen, y evaluando sólo las guardas del `ALT` que tengan alguna posibilidad de hacerse ciertas.
- Minimización de la cantidad de procesos existentes simultáneamente en el sistema y de los tiempos de espera para sincronizar, a través de la emisión de interrupciones del proceso controlador del `ALT` a los evaluadores de guardas que los obliga a terminar tan pronto como alguna guarda se ha hecho cierta.

5 Conclusiones

Se desarrolló un compilador que hace dos pasadas, la primera sobre el código fuente Occam generando directamente un árbol correcto, chequeado y optimizado; y la segunda sobre el código intermedio generado por la primera. Se implementaron optimizaciones que influyen tanto en la eficiencia de ejecución del programa Occam como en el uso de recursos provistos por el sistema operativo. Como extensiones a Occam, se definieron canales que comunican a un proceso con la entrada y la salida estándares de UNIX.

References

[1] A. Aho, R. Sethi, J. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986.

[2] P. Ancilotti, M. Boari. *Programmazione Concorrente*. Versione Preliminare, 1987.

[3] M. Bach. *The Design of the UNIX Operating System*. Prentice-Hall Int., 1986.

[4] E. W. Dijkstra. Cooperating Sequential Processes. Technical Report EWD-123, Technological University, Eindhoven, The Netherlands, 1965.

[5] C. Hoare. *Communicating Sequential Processes*. Prentice-Hall Int., 1985.

[6] M. E. C. Hull. Occam - A Programming Language for Multiprocessor Systems. *Computer Languages*, Volume 12, Number 1, pp. 27-37.

[7] INMOS Limited. *OCCAM Programming Manual*. Prentice-Hall Int., 1984.

[8] J. Peterson, A. Silberschatz. *Operating Systems Concepts*. Addison-Wesley, 1985.

[9] W. R. Stevens. *UNIX Network Programming*. Prentice-Hall Int., 1990.

Un Nuevo Paradigma de Enseñanza para la Disciplina de Computación

Juan Alvarez Rubio
Departamento de Ciencias de la Computación
Universidad de Chile
Blanco Encalada 2120, Santiago, Chile
uucp: jalvarez@dcc.uchile.cl - BITNET: jalvarez@uchcecvm

Resumen

La reciente definición de la computación como disciplina científica, matemática y de ingeniería ha inducido un nuevo paradigma de enseñanza. El anterior énfasis en la programación de computadores está siendo reemplazado por un tratamiento general de todas las subáreas temáticas de la disciplina. Este nuevo enfoque se está aplicando en la formación de especialistas, de ingenieros de otras áreas, y en la educación secundaria.

Keywords: Educación en Computación, Informática Educativa, Curriculum ACM-IEEE/CS.

1 Introducción

A fines de 1988 se produjo un hito histórico en el área de Computación. Un grupo de tarea conjunto de las sociedades científicas americanas más importantes del área, la ACM (Association for Computing Machinery) y la Sociedad de Computación de la IEEE (Institute of Electric and Electronic Engineers), publicó el informe *Computing as a Discipline* [10] que, después de más de 40 años de investigación y desarrollo informático, propone un marco que intenta definir y delimitar el quehacer computacional.

En primer lugar, la computación se considera ahora una **disciplina**, por cuánto sus dimensiones teóricas, de abstracción o modelamiento, y de diseño provienen de las **matemáticas**, de las ciencias experimentales, y de la ingeniería, respectivamente. De esta manera, se supera la polémica respecto de intrepretar y reducir la computación a uno sólo de los aspectos anteriores.

En segundo lugar, se propone una breve definición de la disciplina al considerarla "el estudio sistemático de los procesos algorítmicos que describen y transforman información: su teoría, análisis, diseño, eficiencia, implementación, y aplicación". La pregunta fundamental que subyace en la computación es ¿Qué podemos automatizar (eficientemente)?.

Finalmente, se plantea una subdivisión de la disciplina en nueve subáreas temáticas, cada una con sus propias bases teóricas, sus propios procesos de diseño y abstracción, y una comunidad de investigadores y publicaciones: Algoritmos y Estructuras de Datos, Lenguajes de Programación, Arquitectura, Computación Numérica y Simbólica, Sistemas Operativos, Ingeniería y Metodología de Software, Sistemas de Bases de Datos y de Recuperación de Información, Inteligencia Artificial y Robótica, y, Comunicación Humano-Computador (que incluye Computación Gráfica e Interfaces con el Usuario).

2 Enseñanza de la Disciplina de Computación

El marco anterior sugiere un cambio significativo del paradigma de enseñanza. El antiguo enfoque que se iniciaba y centraba en la programación, debe ser reemplazado por un tratamiento horizontal que, introduciendo todas las subáreas temáticas, permita obtener una visión panorámica de la disciplina. La verticalidad o profundización se reserva a cursos especializados más avanzados.

El esquema anterior, si bien estuvo motivado por la necesaria renovación de los programas de estudios de los especialistas, sin embargo puede extenderse a distintas audiencias, haciendo las adaptaciones que resulten pertinentes. A continuación revisaremos la situación en diferentes contextos educacionales.

2.1 Especialistas en computación

2.1.1 Curriculum 1991

En el informe *Computing Curricula 1991* [5] se propone un nuevo marco para la definición de programas de pregrado en computación. A la subdivisión de nueve sub-áreas, se agregó el contexto social y profesional de la disciplina. Adicionalmente se distinguen doce conceptos "recurrentes", que están presentes tanto en las sub-áreas como en las dimensiones teórica, de abstración y de diseño: asociación (binding), complejidad de problemas grandes, modelos conceptuales y formales, consistencia y completitud, eficiencia, evolución, niveles de abstracción, ordenamiento espacial, ordenamiento temporal, reusabilidad, seguridad, balances y consecuencias.

Para efectos de la estructuración curricular se define un conjunto de 55 unidades de conocimiento que subdividen los contenidos de las nueve (más un) subáreas temáticas, conformando un núcleo mínimo común a todos los programas de pregrado. La idea es que estas unidades sirvan como base para estructurar cursos específicos, combinándolas y orientándolas de acuerdo a los objetivos, prioridades y restricciones de cada institución educacional. Por supuesto, este núcleo debe ser complementado con requerimientos de matemáticas, ciencias, y material adicional y suplementario de computación.

Los lineamientos generales recomiendan que los programas de pregrado aseguren tanto amplitud como profundidad. Amplitud, en el sentido de cubrir todas las áreas temáticas. La profundidad se debe reflejar enfatizando aquellas áreas y procesos acordes con la orientación y especialización del programa. Por su parte, el rol de la programación, no se reduce a la codificación en un determinado lenguaje o arquitectura, sino que se amplía a todas las actividades que involucra la solución algorítmica de problemas, presente en todas las sub-áreas. Adicionalmente, y en consistencia con la dimensión científica de la disciplina, se recomienda un novedoso enfoque para el trabajo de laboratorios. Al respecto, se sugieren actividades de laboratorio tanto abiertas como cerradas. Estas últimas de naturaleza semejante al trabajo práctico y experimental de otras especialidades, con sesiones guiadas de realización de trabajos específicos.

2.1.2 Cursos introductorios

Resulta particularmente importante el diseño del primer curso para los especialistas en computación. Considerando que la propuesta curricular anterior permite diversas alternativas, es conveniente revisar algunas implementaciones y proposiciones específicas.

El curso *Foundations of Computer Science* de la State University of New York at Stony Brook [12] está diseñado para superar la debilidad de los alumnos en matemáticas discretas y en resolución de problemas, y para ilustrar los fundamentos matemáticos de la computación. Por ejemplo, los principios de la lógica se usan para entender la programación lógica. Por su parte, las funciones son la teoría que subyace en los lenguajes aplicativos o funcionales. El aprendizaje es reforzado a través de módulos de laboratorio computacionales.

En la misma línea anterior, David Gries propone mezclar en un curso de dos semestres el primer curso de programación y el segundo de matemáticas discretas [11]. De esta manera en cada etapa se enseña la teoría y la aplicación correspondiente. Concretamente se enseñarían los estilos funcional y algorítmico de programación. Primero el funcional, pasando de la inducción matemática

a la manipulación de funciones recursivas. El enfoque imperativo entraría en escena al tratar la economía de tiempo y espacio.

2.2 Ingenieros de otras especialidades

El nuevo marco, representado tanto por la definición de la disciplina como la proposición curricular, está influenciando y modificando la enseñanza de la computación en los distintos niveles y para diferentes audiencias. En primer lugar, se están rediseñando los primeros cursos para los estudiantes de ingeniería (especialistas y no-especialistas), de modo de cambiar el énfasis en programación hacia la introducción de los conceptos fundamentales de cada una de las áreas temáticas [6].

Recomendaciones específicas sugieren un primer curso anual de *Introducción a la Programación y al Uso de Herramientas de Software* como una manera de presentar la disciplina y nivelar los eventuales conocimientos previos [8]. Cabe señalar que el contexto para tratar las herramientas es introducir las subáreas de Comunicación Humano-Computador, Sistemas de Bases de Datos, e Ingeniería de Software. Un segundo curso semestral [7] permite concluir el tratamiento de las subáreas temáticas.

En resumen, se apunta a entregar a los estudiantes de ingeniería una visión general y horizontal de la computación que permita comprender y asimilar los progresos tecnológicos presentes y futuros. La ventaja de este esquema es que puede ser completamente compartido con estudiantes de la especialidad de computación, los que profundizarán sus conocimientos en cursos más avanzados.

2.3 Enseñanza secundaria

2.3.1 Proposición de un Nuevo Curso

Recientemente, en noviembre de 1991, un grupo de tarea del consejo de educación de la ACM publicó un borrador de una proposición de un curso anual de computación para la educación secundaria [3]. Su alcance, profundidad, amplitud y metodología es similar a los cursos secundarios de otras asignaturas científicas, y consistentemente está destinado a todos los estudiantes, y no sólo a los interesados en carreras especializadas. Este esfuerzo complementa, actualiza, y proporciona otro enfoque, a la proposición previa de 1985 centrada en programación y las aplicaciones e implicaciones del uso de los computadores [1].

El curso propuesto sigue los lineamientos más recientes de la disciplina, incluyendo todas las subáreas temáticas organizadas de manera de adecuarse a estudiantes secundarios. La metodología

sigue el modelo de la enseñanza general de otras disciplinas científicas, basándose principalmente en laboratorios cerrados usando computadores y ejercicios a ser desarrollados sin computadores. Para el trabajo supervisado en laboratorio se deben diseñar guías que, teniendo objetivos y contenidos específicos, permitan a los estudiantes formular modelos (abstracciones) y entender la teoría subyacente. Los alumnos deben informar sus observaciones entregando informes escritos. Adicionalmente, se deben plantear ejercicios y proyectos que puedan ser desarrollados por los propios alumnos fuera de las horas de clases.

2.3.2 Preparación de Profesores

Las proposiciones curriculares para la preparación de profesores de computación en el nivel medio han evolucionado de acuerdo a las proposiciones de cursos para los alumnos. Así, la proposición de ACM para la certificación de profesores que pudieran enseñar los cursos propuestos en el año 1985 [2] se centraba consistentemente en la programación. Una posterior actualización de la propuesta anterior [4] se planteó explicitamente con una orientación hacia los fundamentos de la ciencia de la computación.

Una proposición que asimila tanto la nueva definición de la disciplina de computación, como la nueva proposición de curso para los estudiantes, recomienda al menos cuatro cursos en los programas de postgrado en Informática Educativa [9]: un curso de Introducción a la Resolución de Problemas y a la Programación; dos cursos de Introducción a las Subáreas de la Disciplina de Computación; un curso para tratar el Contexto Social, Etico y Profesional.

3 Conclusiones

Un nuevo paradigma de enseñanza de computación se ha consolidado en los últimos años. Derivado de la reciente definición de la computación como disciplina, su propósito es proporcionar una visión general de todas las subáreas temáticas, reemplazando el énfasis anterior en programación de computadores. Este nuevo enfoque se está aplicando en diversos contextos educacionales: formación de especialistas, enseñanza de computación para ingenieros, alumnos y profesores de computación en la enseñanza secundaria.

Respecto de la formación de especialistas, existe un documento con recomendaciones para diseñar curriculums que se ajusten a estas nuevas ideas. Por su parte, los primeros cursos están siendo diseñados de manera que integren contenidos de matemáticas discretas y de computación.

La enseñanza de computación para estudiantes de ingeniería, si bien puede coincidir completamente con los cursos introductorios para los especialistas, pretenden entregar una visión general y horizontal de la disciplina.

A nivel de la educación secundaria, se pretende enseñar computación siguiendo el modelo de otras asignaturas científicas. El tratamiento panorámico global de la disciplina se basa en la realización de laboratorios computacionales cerrados y ejercicios y proyectos abiertos de mayor envergadura. Consecuentemente, es necesario renovar los planes de formación y actualización de los profesores.

En síntesis, el nuevo paradigma de enseñanza de la disciplina de computación comprende una perspectiva globalizadora que apunta a enseñar y aprender los conceptos fundamentales de todas las subáreas temáticas. De esta manera se pretende satisfacer el desafío de asimilar los desarrollos tecnológicos presentes y futuros del área.

Referencias

[1] Task Force on Curriculum for Secondary School Computer Science; *Computer Science for Secondary Scool: Course Content;* Communications of the ACM, Volume 28 Number 3, 270-274;,March 1985

[2] Task Force on Teacher Certification in Computer Science; *Proposed Curriculum for Programs Leading to Teacher Certification in Computer Science;* Communications of the ACM, Volume 28 Number 3, 275-279; March 1985

[3] Task Force of the Pre-College Subcommittee of the Education Board of the ACM; *ACM Model High School Computer Science Curriculum;* Draft Report; November 1991

[4] ACM/IEEE Task Force on Teacher Retraining; *Retraining Teachers to Teach High School Computer Science;* Communications of the ACM, Volume 31 Number 7, 912-917; July 1988

[5] ACM / IEEE-CS Joint Curriculum Task Force; *Computing Curricula 1991;* ACM Press; 1991

[6] Alvarez Rubio, Juan; *Computación en Licenciaturas en Ingenierías;* I Encuentro Interamericano de Educación en Ingeniería; Puebla, México; Agosto 1990

[7] Alvarez Rubio, Juan; *Un Curso de Introducción a las Sub-Areas de la Disciplina de Computación;* I Congreso Iberoamericano de Educación Superior en Computación; Santiago, Chile; Octubre 1991

[8] Alvarez Rubio, Juan; *A First Computing Course Based on Curricula 1991;* Proceedings 23rd SIGCSE Technical Symposium; Kansas City, USA; March 1992

[9] Alvarez Rubio, Juan; *Cursos de la Disciplina de Computación en Postgrados de Informática Educativa;* Congreso Iberoamericano de Informática Educativa; Santo Domingo, República Dominicana; Junio 1992

[10] Denning, P.J., Comer, D.E.; Gries, D.; Mulder, M.C.; Tucker, A.; Turner, A.J.; Young, P.R.; *Report of the ACM Task Force on the Core of Computer Science;* ACM Press; August 1988

[11] Gries, David; *Teaching Calculation and Discrimination: A More Effective Curriculum;* Communications of the ACM, Volume 34 Number 3, 44-55; March 1991

[12] Henderson, Peter B.; *Discrete Mathematics as a Precursor to Programming;* Proceedings 21st SIGCSE Technical Symposium; Feb 1990

TÉCNICAS DE PROCESSAMENTO PARALELO PARA SISTEMAS GRÁFICOS DE ALTO DESEMPENHO

Manuel Lois Anido

Núcleo de Computacão Eletrônica da UFRJ
Cx. Postal 2324 - CEP 20.001 - Rio de Janeiro - Brasil
e-mail: ncd10121@ufrj.bitnet

Resumo

Este artigo descreve as principais técnicas de processamento paralelo empregadas na geração de imagens por computador e apresenta as arquiteturas de alguns sistemas de visualização de alto desempenho. Inicialmente este artigo analisa os problemas envolvidos na geração de imagens, de forma qualitativa e quantitativa. Em seguida é apresentada uma taxonomia de sistemas de geração de imagens, discutindo as principais vantagens e restrições de cada um.

Palavras chave: Computação Gráfica, Processamento Paralelo, Geração de Imagens, Arquiteturas Gráficas.

1. Introdução

Tem-se conseguido grande progresso no campo de geração de imagens por computador nas duas últimas décadas. A maioria dos avanços tem sido nos campos de síntese e de geração de imagens. O desenvolvimento destes campos tem tido lugar em pelo menos duas principais frentes. Uma direção de pesquisa tem sido melhorar o realismo das cenas mostradas, independente do tempo necessário para criar a imagem. A outra principal direção tem sido construir sistemas altamente interativos, que devem produzir imagens sucessivas dentro de um período fixo de tempo (usualmente 1/30 do segundo, ou menos), ou seja, que devem operar em tempo real.

Sistemas de visualização tridimensional (3D) tem sido utilizados em sua grande maioria em aplicações de simuladores de vôo. Entretanto, como resultado dos avanços da tecnologia VLSI (Very Large Scale Integration), outros campos de aplicação estão sendo desenvolvidos. Estes campos incluem: visualização científica interativa, películas de cinema, animação por computador, análise de movimento, simulação de tráfego, videogames, etc. Estas novas aplicações tem aumentado a demanda por sistemas capazes de produzir imagens de alta qualidade e resposta rápida aos comandos do usuário.

2. Conceituação do Problema

2.1. Descrição Formal dos Objetos

Imagens são produzidas por um conjunto de transformações aplicadas a uma representação 3D de modelos de objetos armazenada em disco. O processo de geração de imagens consiste na criação de uma imagem 2D, definida pela cor e luminancia dos pixels no espaço da imagem formada pelo tubo de raios catódicos. Modelos para GIC (Geração de Imagens por Computador) concentram-se na estrutura das cenas, na sua forma, e na relação espacial. Diferentes formas de representação podem ser usadas para o modelo tridimensional.

Um modelo na forma geométrica computacional é representado por um conjunto de objetos sólidos definidos por expressões matemáticas, que incluem superfícies bicúbicas paramétricas, B-splines e curvas de Bezier [1]. Entretanto, a descrição de superfícies em forma matemática conduz a uma enorme carga computacional. De forma a facililtar a construção de um modelo complexo, pode ser usado um esquema de Geometria de Construção de Sólidos (CSG), o qual explora a noção de *adicionar* e *subtrair* blocos sólidos simples. Esta representação tem muitas vantagens, particularmente durante a fase de construção, mas é rígida com relação às transformações geométricas necessárias para criar vistas do modelo, dentro das limitações de tempo real.

A técnica consagrada de representação do modelo, para atender às restrições das transformações em tempo real, consiste em definir o modelo indiretamente, através de conjuntos de faces que são representadas por conjuntos de arestas. Estas faces podem ter a forma de polígonos, usualmente triangulos ou quadriláteros. Esta representação é usualmente conhecida como B-rep [1].

Fractais são uma outra forma de modelamento de objetos e tem recebido muita atenção da comunidade científica nos últimos anos [2]. As imagens resultantes desta técnica são espetaculares, e muitos métodos de gerar fractais tem sido desenvolvidos. O termo *fractal* tem sido generalizado pela comunidade científica para incluir objetos fora da definição original de

Mandelbrot. Esta forma de modelamento também leva a uma elevada carga computacional, podendo requerer horas de processamento para gerar uma imagem.

2.2 Técnicas de Visualização

A. Método Projetivo com Pipeline Gráfica

A técnica de visualização mais comum é o *método projetivo*, no qual os componentes da cena são submetidos a transformações geométricas (para transladá-los ou mudar o ponto de vista) e em seguida são projetados na tela. Este é o método utilizado em sistemas gráficos que operam em tempo real, onde um certo grau de realismo é sacrificado em prol de interatividade.

Uma organização em pipeline é uma estrutura natural para geração de imagens porque cada estágio geralmente só depende dos resultados do estágio anterior. Além disso tal organização diminui a alta demanda de processamento, pois diversas tarefas podem ser executadas simultaneamente, embora não necessáriamente na mesma imagem. A figura 1 ilustra um pipeline típico para geração de imagens que emprega primitivas (linhas e polígonos) e técnicas de sombreamento convencionais. Esta subdivisão do pipeline é lógica e não física, pois alguns estágios do pipeline podem ser implementados tanto em software como em hardware.

B. Ray-Casting e Ray-Tracing

Ray-Casting e Ray-Tracing [3] são técnicas que consistem em determinar a visibilidade de superfícies, através da emissão de raios imaginários de luz, partindo do olho do observador até os objetos da cena. A interseção dos raios com a tela especifica os pontos de formação da imagem. O método Ray-Tracing é usualmente uma extensão do algoritmo de ray-casting, para incluir reflexões, translucencia, sombras e refrações e apresenta bons resultados para luz especular. A cor de cada pixel é determinada pela contribuiçao de todos os raios.

Estes métodos propicíam imagens altamente realísticas, porém o número de operações envolvidas é extremamente elevado, fazendo com que as imagens levem horas para serem geradas nos computadores atuais. Uma tela de 1024x1024 pixels requer mais de um milhão de operações de cálculo de interseções raio-objetos, que por si só são operações sofisticadas. A carga de processamento cresce quadráticamente com o número de pixels em uma das laterais do CRT ($O(N^2)$). Por outro lado, o método é favorável à utilização de processamento altamente paralelo, permitindo diversos níveis de paralelismo.

C. Radiosidade

A técnica de radiosidade surgiu a partir dos modelos de transferência de calor e, ao contrário de ray-tracing, apresenta bons resultados para luz difusa, embora demande elevada capacidade de processamento. Diferentemente de Ray-Tracing, a carga computacional cresce quadráticamente ($O(N^2)$) com o número de objetos, ao invés do número de pixels por objeto. As superfícies podem ser refletoras ou emissoras de energia. O objetivo do método é calcular o balanço de energia do sistema.

2.3 A Imagem e a Memória de Vídeo (Frame Buffer)

Uma imagem pode ser imaginada como uma matriz tridimensional, onde linhas e colunas correspondem aos eixos x e y da tela e a terceira dimensão (z) corresponde às características de cada pixel, ou seja, R,G,B, profundidade, intensidade, etc. Os eixos x e y são normalmente mapeados numa memória de vídeo, enquanto o eixo z é normalmente mapeado em diversos planos de memória.

Nos sistemas que utilizam uma memória para armazenar a imagem mostrada ("frame buffer"), são normalmente empregadas memórias de vídeo ("video RAMs" - VRAM), projetadas especificamente para permitir que a saída de vídeo seja independente de outras operações no frame buffer. Um chip VRAM é similar a um chip DRAM convencional, mas contém um registro de deslocamento com entrada paralela e saída serial, conectado a uma segunda porta de dados, por onde saem os bits em série que irão endereçar a tabela de cores.

2.4 Principais Barreiras ao Desempenho

Sistemas gráficos podem ser construídos com os mais diversos desempenhos. No entanto, para projetar arquiteturas gráficas de alto desempenho, exisem três barreiras importantes que devem ser ultrapassadas, a saber:

a. *Processamento Geométrico:* Acelerar transformações, recorte e perspectiva, além dos limites possíveis com um só processador de ponto flutuante (ou ponto fixo).

b. *Processamento Inteiro de Pixels*: Acelerar o preenchimento de polígonos e o processamento de pixels, além da taxa obtenível com um simples processador gráfico.

c. *Banda Passande do Frame-Buffer*: Prover leituras e escritas no frame-buffer mais rápidas

do que as suportadas por um sistema de memória convencional.

2.5 Demanda Computacional

Consideremos uma cena sofisticada composta de 10.000 polígonos e uma taxa de atualização de 20 cenas por segundo (a nave espacial ENTERPRISE do filme Guerra nas Estrelas foi formada com 450.000 polígonos !!).

O número total de instruções de ponto flutuante necessárias para realizar as operações geométricas é de aproximadamente 40 milhões [1]. Nas operações de rastreio e acesso ao "frame buffer", é necessário realizar 65 milhões de somas e 102 milhões de acessos ao "frame-buffer", por segundo [1].

Da estimativa de carga computacional acima é possível concluir que é mandatório o uso de processamento paralelo e que é necessária uma organização adequada do frame-buffer para atender tal formidável demanda computacional.

3. Arquiteturas Para Computação Gráfica Usando Processamento Paralelo

3.1. Introdução

Sistemas gráficos de alto desempenho geralmente possuem arquiteturas com múltiplos processadores, grupados em três grandes blocos funcionais, que são: sistema de acesso ao banco de dados e de percurso da lista de polígonos, sistema de operações geométricas, e sistema de rasterização. É usual dizer que o sistema de operações geométricas constitui o 'front-end' do sistema gráfico e que o sistema de rasterização constitui o 'back-end'. Em sistemas que realizam a simulação do comportamento de aparelhos complexos, como aviões (simuladores de vôo) ou automóveis, normalmente é utilizado um processador adicional dedicado a tal tarefa, o qual transmite informações ao resto do sistema gráfico.

3.2. Estratégias Típicas de Paralelização em Computação Gráfica

As abordagens típicas para paralelizar são:

- *Paralelização no Espaço dos Objetos:*
  - Objetos são processados em paralelo;
  - Pixels sao gerados em sequência;
  - O aumento do número de objetos implica no aumento do número de processadore
  - O aumento do número de pixels implica em processadores mais rápidos;
  - No caso extremo tem-se 1 processador por objeto;
- *Paralelização no espaço da Imagem:*
  - Objetos são processados em sequência;
  - Pixels são processados em paralelo;
  - O aumento do número de objetos implica em processadores mais rápidos
  - O aumento do número de pixels implica no aumento do número de processadores;
  - No caso extremo tem-se 1 processador por pixel;
- *Paralelização Funcional*
  - Objetos são processados em paralel ou série;
  - Processadores alocados por tarefa;
  - Pipeline gráfica;

4. Arquiteturas para Operações Geométricas com Múltiplos Processadores

Num sistema gráfico que transforma objetos do espaço tridimensional para o espaço da tela, uma possível sequência de operações para alcançar as transformações necessárias é:

- Remoção de faces de costas para o observador;
- Translação, Rotação, e Escalamento;
- Recorte;
- Projeção Perspectiva;
- Iluminação dos Polígonos;

4.1. Organização em Pipeline (ou Paralelismo Horizontal)

Um dos mais conhecidos pipelines gráficos para operações geométricas foi introduzido por Clark [4], denominado *geometry engine*, o qual é ilustrado na figura 2. O *geometry engine* original consiste de um pipeline de 12 circuitos integrados microprogramados identicos que podem realizar funções distintas, de acordo com sua posição no pipeline. Uma versão mais recente deste pipeline (com 5 circuitos integrados) é utilizada na última linha de sistemas gráficos da Silicon Graphics.

Este tipo de solução com circuitos integrados especializados microprogramados é eficiente, mas não apresenta flexibilidade suficiente para permitir a inclusão de funções adicionais. Um outro cuidado adicional que deve ser tomado neste tipo de organização é fazer com que haja um balanceamento de carga entre os diversos estágios do pipeline.

Um outro exemplo empregando pipeline é descrito por Atkin e Ghee [5]. Neste sistema, é empregado um pipeline de transputers T800 para realizar as operações geométricas, conforme ilustrado na figura 6. O desempenho mencionado no artigo é baixo (quando

comparado a sistemas comerciais) em virtude da utilização dos canais seriais para passar informação de um estágio do pipeline para outro e também em função de ser um sistema simples de teste. Entretanto, o artigo serve para mostrar que modificações radicais devem ser feitas na arquitetura de pipeline de transputers para conseguir alto desempenho em operações gráficas. Por outro lado, empregando uma metodologia Ray-Tracing, ao invés de malhas de polígonos, é possível conseguir alto desempenho (quando comparado cou outras arquiteturas) empregando uma organização adequada de transputers. Isto se deve ao fácil mapeamento do problema numa rede de transputers, onde o tempo de processamento é muito maior que o tempo de comunicação.

Vale lembrar que uma solução que empregue simplesmente pipeline no 'front-end' tem limitações para obter elevado desempenho, pois o processador alocado para percorrer a lista de polígonos ('display list') provavelmente não conseguirá suprir o resto do pipeline. Portanto uma solução paralela poderá ser necessária.

4.2. Organização em Paralelo (ou Paralelismo Vertical)

A utilização de imagens descritas por malhas de polígonos permite repartir a carga de trabalho entre diversos processadores operando concorrentemente, pois em geral não há interdependencia nas operações geométricas entre as diversas primitivas gráficas. Tendo por base este fato, diversas arquiteturas paralelas tem sido propostas e utilizadas comercialmente.

O sistema visual IMAGE IV do simulador de vôo da Singer Link-Miles utiliza diversos processadores do tipo DSP (Digital Signal Processor) operando em paralelo, organizados conforme ilustra a figura 6. Os canais DMA controlam a transferencia dos dados de entrada e de saída dos DSPs. O uso deste tipo de processadores nesta aplicação deve-se ao fato de os mesmos possuírem operações aritméticas rápidas, que é um requisito fundamental para acelerar as operações geométricas.

Foi desenvolvido, na Universidade de Southampton, (UK) um sistema visual de baixo custo que utiliza até 30 microprocessadores MC68000-12MHz operando em paralelo, com uma organização semelhante à do sistema IMAGE IV [6]. As principais diferenças entre este sistema e o sistema IMAGE IV estão no fato de o sistema de Southampton também realizar o preenchimento de polígonos, além transmitir os dados de saída numa forma codificada e fácil de ser interpretada pelo controlador da memória de video para preenchimento da memória.

Com o intuito de aumentar o desempenho do sistema gráfico de Southampton em operações geométricas e também reduzir o número de processadores necessários, foi desenvolvida uma nova arquitetura utilizando processadores RISC especialmente projetados para esta aplicação [7,8]. Diversos processadores RISC são organizados em paralelo (em grupos de 4), conforme ilustrado na figura 7. Cada processador RIG é capaz de operar a 30 MIPS (pico), possuindo buffers de entrada e saída embutidos na área de registros gerais do sistema, para acelerar a transferencia de dados de E/S. Canais DMA de entrada e saída coordenam a transferência de dados do sistema. A transferência de dados é feita por acesso direto à memória pois o tempo de processamento é da mesma ordem de grandeza do tempo de transferência. O sistema é organizado de tal forma que o balanceamento de carga pode ser fàcilmente distribuído. A ordem dos pacotes de dados de entrada é monitorada e passada ao sistema de rasterização de polígonos que irá recompo-la, caso necessário.

Torborg descreve uma arquitetura para operações geométricas utilizando processadores de 32 bits microprogramados operando em paralelo [9]. Cada processador opera a 12.5 MIPs e 25 MFLOPs. Esta arquitetura foi especialmente projetada para executar o conjunto de comandos do standard para gráficos 3D - PHIGS+. A organização geral desta arquitetura assemelha-se às outras arquiteturas paralelas descritas nesta seção, com um barramento de entrada de dados 3D e comandos e um barramento de saída para dados 2D e comandos para o controlador da memória de tela.

4.3. Organização com Pipelines em Paralelo (Paralelismo H & V)

Grimdale e seu grupo [10] apresentaram uma proposta de arquitetura que consiste de circuitos integrados funcionalmente semelhantes ao *geometry engine*, organizados em diversos pipelines em paralelo, visando obter maior desempenho nas operações geométricas. Um pipeline completo é composto de 16 circuitos integrados, e o objetivo do grupo de Sussex (UK) é conseguir um desempenho superior a 10.000 polígonos em tempo real.

5. Arquiteturas Para Geração da Imagem com Múltiplos Processadores

O subsistema de rasterização ('back-end') cria a imagem final através da conversão do conjunto de primitivas passadas pelo subsistema de operações geométricas. As principais operações necessárias neste estágio são:

- Determinar a visibilidade de cada pixel de uma primitiva (polígono ou objeto);
- Determinar o sombreamento de cada pixel;
- Acrescentar funções especiais como por exemplo texturas;

O artigo de Gharachorloo, Gupta, Sproull e Sutherland [11] apresenta uma excelente caracterização das principais técnicas de rasterização.

5.1. Arquiteturas com Pipeline para Geração de Imagens

5.1.1. Pipeline por Ordem de Objetos

Esta forma de organização é um método direto de prover concorrencia aos cálculos de rasterização, distribuindo os vários passos de um algoritmo de software, num pipeline de hardware. Esta técnica pode ser usada com os dois principais métodos de rasterização: ordem de objetos (algoritmos z-buffer, depth-sort, e BSP-tree) e ordem de imagem (algoritmos de linhas de vídeo - 'scan-line'). Vale notar que na técnica por ordem de objeto, um determinado objeto é todo rasterizado antes de passar para o próximo objeto, mesmo que o próximo objeto seja interceptado por linhas de vídeo comuns a ambos.

Os principais blocos deste pipeline, que geralmente tem um processador dedicado, executam:

1. Processamento de Polígono - Determinar a linha de vídeo inicial interceptada pelo polígono e calcular a inclinação dos lados (ou calcular os valores *Delta* para x,z,R,G,B para cada lado).

2. Processamento de Lados - As linhas de vídeo dentro de cada primitiva são processadas uma por uma. Os valores *Delta* calculados no estágio anterior são usados para calcular x,z,R,G,B, na interseção dos lados com a linha de vídeo. Os valores *Delta* para incrementar z,R,G, e B de pixel para pixel dentro de um span (sequência contígua de pixels) são então calculados.

3. Processamento do Span - Para cada span, usando o *Delta* calculado anteriormente, calcular os valores x,z,R,G,B para os pixels dentro do .span. O sombreamento Gourad [12] também pode ser feito neste bloco do pipeline. O valor da profundidade z (que indica se o pixel será ocultado) é comparado com o z anterior armazenado naquela posição e caso seja menor (novo mais próximo do observador), o novo pixel substitui o velho.

Este algoritmo só é valido para polígonos convexos. O tempo de processamento do método por ordem de objeto tende a crescer com o número de objetos na cena, sejam eles visíveis ou não.

5.1.2. Pipeline por Ordem de Imagem

Neste tipo de método, a imagem é calculada linha de vídeo por linha de vídeo (ou pixel por pixel), ao invés de primitiva por primitiva. Para evitar primitivas que não foram transformadas para o espaço da tela, primitivas são ordenadas em caixas, de acordo com a primeira linha de vídeo em que aparecem. Os principais blocos do pipeline por ordem de imagem (que geralmente possuem um processador dedicado) são:

a. Ordenador em Y - Coloca cada lado do polígono na caixa correspondente à linha de vídeo em que ele aparece primeiro.

b. Ordenador de segmentos ativos - Le lados das caixas mantendo uma tabela de lados ativos para a linha de vídeo corrente. Desta tabela ele constrói uma lista de segmentos ativos (span dentro de um polígono), o qualé ordenado pelo valor *x* do extremo esquerdo de cada segmento.

c. Gerador de Spans Visíveis - Percorre a lista de segmentos ativos, comparando os valores *z* onde necessário, e dá saída da sequência de spans visíveis na linha de vídeo corrente.

d. Sombreador - Realiza o sombreamento Gourad ou Phong, calculando a intensidade nos pixels do span.

Deve-se notar que não é necessária memória de vídeo neste tipo de sistema, desde que o sistema gráfico consiga gerar pixels à taxa do vídeo. Isto é muito difícil atualmente devido ao grande número de objetos e ao número de pixels dos sistemas atuais, que demandam uma enorme banda passante do vídeo. No entanto, este método foi a base de muitos sistemas gráficos dos simuladores de vôo de Evans & Sutherland nos anos 70.

Este tipo de método é limitado pela linha de vídeo mais ocupada (com maior quantidade de spans), o que limita a complexidade da cena. O tempo de processamento tende a crescer com a complexidade das partes visíveis da imagem, ao invés do número de objetos.

5.2. Arquiteturas Paralelas para Geração de Imagens

Paralelismo de imagem é atrativo porque os pixels da tela podem ser gerados em paralelo de diversas formas. As formas mais comuns de subdividir a tela são: em filas, em colunas, e entrelaçada. Um sistema que gera a imagem em paralelo, rasteriza em ordem de objetos e portanto uma memória de vídeo (frame buffer) é necessária para armazenar os resultados intermediários.

5.2.1. Arquiteturas com Memória Particionada

A idéia básica desta forma de ataque do problema é associar um processador a cada partição da memória de vídeo. Esta organização permite computar regiões em paralelo, aumentando o desempenho do sistema gráfico. A banda passante da memória de vídeo pode ser aumentada, provendo um canal de acesso separado à partição para cada processador. As duas formas mais comuns de subdividir a memória de vídeo são: partição

contígua e partição entrelaçada, as quais são ilustradas na figura 3.

A principal vantagem da partição contígua é que as primitivas só tem que ser processadas nas regiões em que podem ser visíveis, e estas regiões podem ser determinadas rapidamente. Como principal desvantagem está o fato de ser mais susceptível a gargalos, os quais podem ocorrer caso todas as primitivas caiam na região de um processador.

A partição entrelaçada foi proposta inicialmente por Fuchs [13] e permite atingir uma melhor divisão da carga de trabalho, pois quase todos os polígonos (exceto os minúsculos) caem em todas as partições da memória de vídeo. Cada processador lida com todas as primitivas, embora só uma fração dos pixels. O mais importante é que o pior caso desta organização depende do numero total de primitivas e não da região mais densa. Esta é a técnica mais utilizada atualmente.

Uma melhoria da partição entrelaçada foi proposta por Clark e Hannah [4], para tomar vantagem de cálculos comuns a múltiplos processadores. Este método é utilizado nos sistemas mais recentes da Silicon Graphics, cuja arquitetura é ilustrada na figura 8.

5.2.2. Memória de Vídeo com Lógica

Neste tipo de método, processamento é incorporado a partições da memória de vídeo, fornecendo novos modos de endereçamento dos pixels, bem como a habilidade de endereçar um retangulo inteiro de pixels da memória de vídeo, em paralelo. Um exemplo desta organização é o *8x8 Display* desenvolvido por Gupta, Sproul e Sutherland.

Henry Fuchs propos e desenvolveu uma arquitetura denominada *Pixel Planes 4* que explora o paralelismo de imagem ao extremo [13]. Nesta arquitetura, um processador muito simples (1 bit) e uma pequena memória (72 bits) são incorporados a cada pixel, formando uma memória de vídeo inteligente. O processador executa a equação básica Ax+By+C, que é a equação de uma reta e reflete os lados dos polígonos. Cada expressão é avaliada em paralelo em todos os pixels da tela, para saber se cada pixel está fora ou dentro do polígono. A figura 9 ilustra a organização básica da arquitetura *Pixel Planes 4*.

5.3. Rasterização por Objeto em Paralelo

Numa arquitetura de rasterização por objeto em paralelo, múltiplas primitivas são processadas em paralelo, de forma que os pixels podem ser gerados mais rápido. O método usual é associar primitivas a um certo número de processadores (idealmente um processador por objeto), cada qual sendo capaz de gerar a imagem completa contendo suas primitivas. Durante a rasterização cada processador de objeto enumera os pixels da tela em ordem de linha de vídeo (em geral), gerando z, R,G e B. As cadeias de pixels de cada processador de objeto tem que ser combinadas para produzir uma única cadeia de pixels para o estágio final. Um dos primeiros sistemas a utilizar este método foi o simulador de vôo General Electric NASA II.

5.3.1. Pipeline de Processador por Primitiva

Um modo simples de combinar as multiplas cadeias de cores de pixels e de valores z, produzidas por múltiplos processadores de objetos é organizar os processadores em pipeline, de forma que uma cadeia de valores de cor e de profundidade z passe através deles (ver figura 4). Cada processador gera valores de cor de de z para sua primitiva no pixel corrente, e compara seu valor z com o valor z do processador anterior no pipeline. Se o valor z de entrada é menor (polígono mais próximo), os valores de cor e z são passados adiante. Se z for maior que o z de entrada, os valores da primitiva são enviados para o estágio seguinte e os valores do estágio anterior são inibidos.

Uma das primeiras arquiteturas projetadas utilizando este princípio foi apresentada por Cohen e Demetrescu. As principais desvantagens deste tipo de arquitetura são o elevado número de processadores necessários num sistema com muitos polígonos e o fato de não utilizar memória de vídeo, a qual é essencial em certas aplicações, como por exemplo a adição de texturas baseadas em mapas. Uma outra desvantagem é o tempo de latencia do pipeline para gerar a imagem. A figura 11 ilustra este tipo de arquitetura.

5.3.2. Árvore de Processador por Primitiva

Este tipo de organização é uma alternativa à organização em pipeline e consiste em usar uma árvore binária com blocos de composição para misturar as cadeias de cor e de coordenadas z. Este método permite que a rasterização nos processadores de objetos seja verdadeiramente síncrona (não há atraso para computar o mesmo pixel), e reduz a latencia ocasionada por um pipeline longo.
A desvantagem é que é necessária lógica especial para realizar a composição.

A primeira arquitetura em árvore com um processador por primitiva foi apresentada por Fussel. Kedem e Ellis [14] desenvolveram a arquitetura denominada Ray Casting Machine, que rasteriza diretamente imagens a partir de uma representação em árvore de Geometria de Construção de Sólidos (CSG).

5.4. Rasterização Híbrida Paralela

Quando as filosofias de paralelismo de objeto e paralelismo de imagem são levadas ao extremo, resultam sistemas pouco eficientes. Isto pode ser

melhorado usando uma combinação das duas técnicas e produzindo sistemas híbridos paralelos, que embora sejam mais complexos que os outros, são mais eficazes.

5.4.1. Buffers Virtuais e Processadores Virtuais

A principal desvantagem das arquiteturas altamente paralelas por objeto e por imagem está na baixa utilização dos elementos processadores (PEs). Uma maneira de aumentar a utilização é construir sòmente uma fração do hardware, mas alocar estes recursos dinamicamente na tela, conforme necessário. Existem duas variantes desta técnica: *buffers virtuais* (para sistemas paralelos por imagem) e *processadores virtuais* (para sistemas paralelos por objetos). De forma similar à memória virtual, ambos os métodos tentam aumentar os recursos físicos aparentes de um sistema, através do remanejamento dinamico de recursos. Um exemplo desta arquitetura é o sistema SAGE (Sistolic Array Graphics Engine).

5.4.2. Arquitetura Paralela com Buffer Virtual

Existe um outro nível de paralelismo em sistemas com Buffer Virtual que usam áreas retangulares e ordenação em caixas completa. Um exemplo deste tipo de arquitetura é o sistema Pixel Planes 5. Similarmente ao sistema Pixel Planes 4, este sistema também possui uma memória de vídeo inteligente, com um processador que permite calcular expressões quadráticas do tipo $Ax+By+Cx+Dx^2+Exy+Fy^2$, que são úteis para desenhar superfífies curvas e o modelo de iluminação de radiosidade esférica.

5.5. Arquiteturas para Ray-Tracing

Conforme já descrito, ray-tracing é um método de visualização poderoso que pode gerar figuras altamente realistas. Infelizmente acarreta uma carga computacional muito grande fazendo com que imagens levem minutos ou horas para serem geradas. Felizmente, algoritmos de ray-tracing podem ser paralelizados de diversos modos e em diversos níveis.

A maioria das arquiteturas para ray-tracing utilizam um grande número de elementos de processamento, numa organização MIMD. Como muitas arquiteturas para ray-tracing assemelham-se a computadores paralelos comerciais, os sistemas mais empregados em geração de imagens ray-tracing são sistemas comerciais paralelos. Entre esses sistemas podemos destacar redes de Transputers (IBM VICTOR, PARSYS SuperNode, MEIKO Computing Surface) , Connection Machine [DELA88], AT&T Pixel Machine [POTM89], iPSC [Intel], e outros mais.

6. Conclusões

Este artigo tentou resumir as principais técnicas e arquiteturas empregadas em sistemas gráficos de alto desempenho. Do exposto ficou patente que a demanda computacional é tão elevada que o uso de pipelining e/ou processamento paralelo são mandatórios para alcançar alto desempenho. Foi também mostrado que é necessária uma elevada banda passante entre a memória de vídeo e a tela, principalemte em monitores de alta resolução.

O aumento do desempenho traduz-se para o usuário em imagens mais realistas (como alto nível de detalhes) e alta interatividade, ou seja, o usuário interage com o sistema gráfico e recebe respostas aos seus comandos em tempo real, tal qual a ação de dirigir um automóvel.

Foi mostrado que ainda há muitas barreiras a serem ultrapassadas, para gerar imagens altamente realistas de forma economica, em tempo real. No entanto, os resultados obtidos nos últimos 20 anos tem sido espetaculares.

7. Referencias

[1] Foley, J.D., Van Dam, A., Feiner, S. and Hughes, J., *Computer Graphics - Principles and Practice,* Addison Wesley, Reading, Mass., 1990.

[2] Mandelbrot, B.B., *The Fractal Geometry of Nature*, Freeman, San Francisco, 1982.

[3] Roth, S. D., *Ray Casting for Modelling Solids,* Computer Graphics and Image Processing, Vol. 18, No. 2, pp. 109-144, February 1982.

[4] Clark, J.H., *Structuring a VLSI System Architecture,* Lambda, Vol.1 No.2, pp.25-30, Second Quarter, 1980.

[5] Atkin, P. and Ghee, S., *A Transputer Based Multi-User Flight Simulator*, Technical note 36, Inmos Ltd., Bristol, UK, 1988.

[6] Allerton, D.J and Zaluska E. J, *A Multi-processor Approach to Image Generation*, IEE Int. Conf. on Simulators, Warwick, England, pp. 226-231, 1986.

[7] Anido, M.L. and Allerton, D.J., *RISC Design for Computer Image Generation*, Microprocessors and Microsystems Journal, Butterworth-Heinemann Publ., Vol. 14, No.6. pp. 341-350, August 1990.

[8] Anido, M.L., *Design and Implementation of a RISC Processor for Computer Image Generation Geometric Computations*, Ph.D. Thesis, Dept. of Electronics & CS, University of Southampton, UK, 1991.

[9] Torborg, J.G., *A Parallel Processor Architecture for Graphics Arithmetic Operations*, SIGGRAPH, vol. 21,

No.4, pp. 197-204, July 1987.

[10] Grimsdale, R.L., *Techniques for Real-Time Image Generation*, Proc. Int. Conf. on Parallel Processing for Computer Vision and Display, Leeds, UK, January, 1988.

[11] Gharachorloo, N, Gupta, S., Sproull R.F. and Sutherland, I.E., *A Characterization of Ten Rasterization Techniques*, SIGGRAPH 89, Boston, August, pp.355-367, 1989.

[12] Gourad, H., *Continuous Shading of Curved Surfaces*,
IEEE Transactions on Computers, Vol.20/6, pp.623-628, June, 1971.

[13] Fuchs, H. and Poulton, J., *Pixel-Planes: a VLSI-Oriented Design for a Raster Graphics Engine*, VLSI Design, Third Quarter, pp. 20-28, 1981.

[14] Kedem, G. and Ellis, J.L., *The Raycasting Machine*, in Proceedings of the 1984 Int. Conf. on Computer Design, pp. 533-538, October 1984.

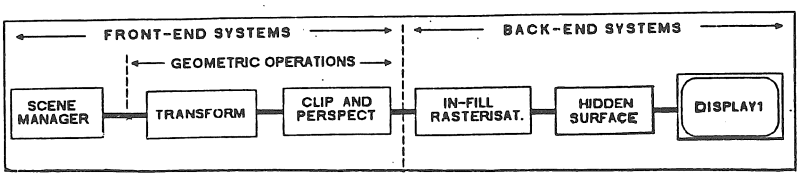


Figura 1 - Pipeline tipica para geração de imagens

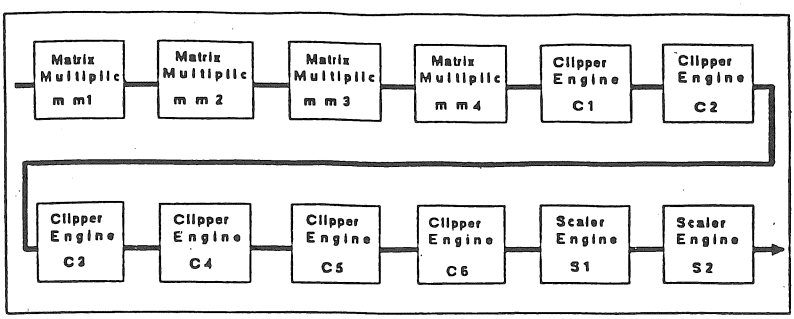


Figura 2 - Elementos do Pipeline do Geometry Engine

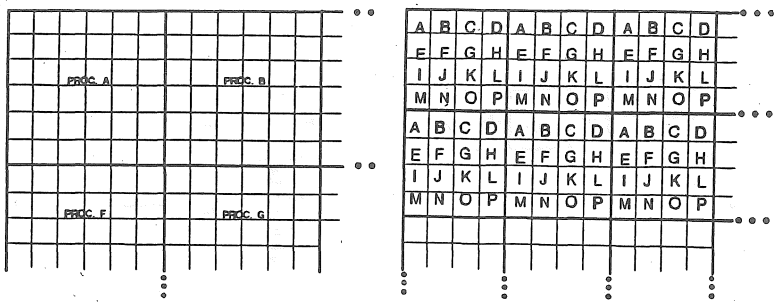


Figura 3 - Partição Contígua e partição entrelaçada da Memória de vídeo

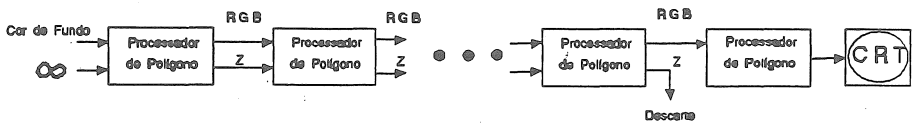


Figura 4 - Arquitetura de Cohen e Demetrescu - pipeline de processador por primitiva

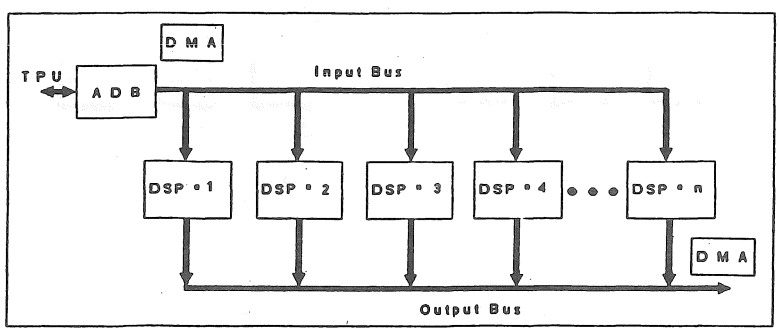


Figura 5 - Paralelismo de DSPs no simulador de Vôo da Singer-Link Miles.

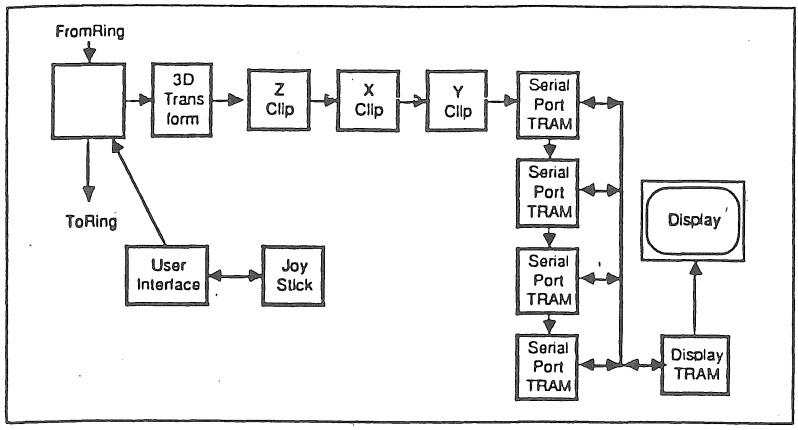


Figura 6 - Sistema Gráfico com Pipeline de Transputers

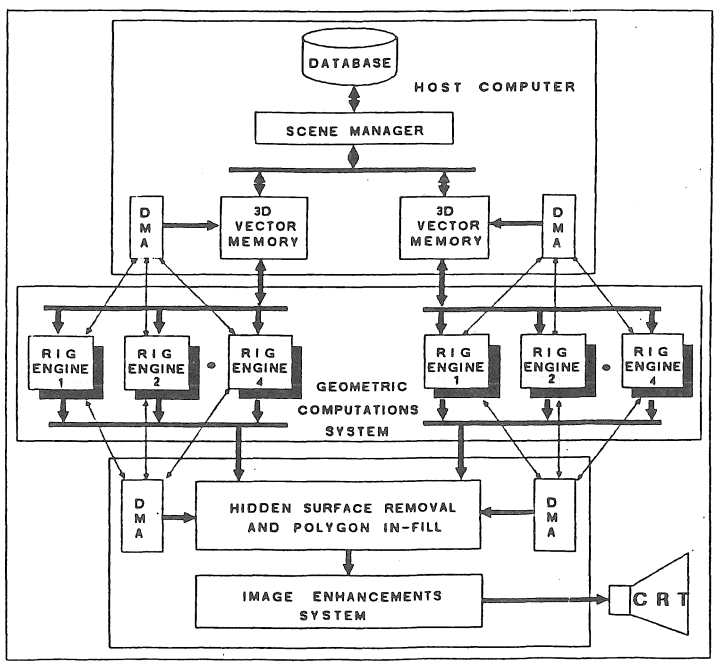


Figura 7 - Sistema MIGS com Processadores RISC Especializados para Operações Geométricas

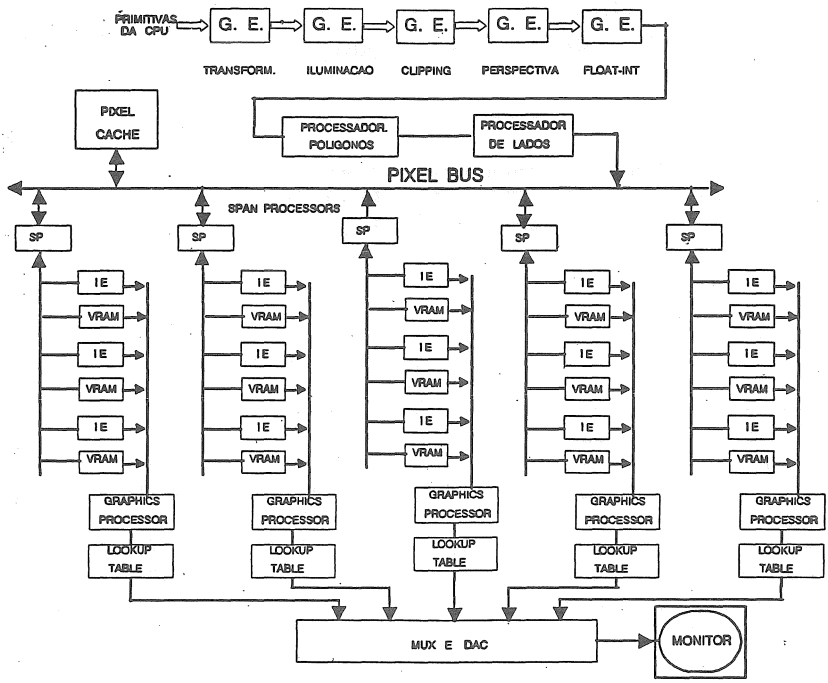


Figura 8 - Arquitetura do Sistema Silicon Graphics 4D-240/GTX

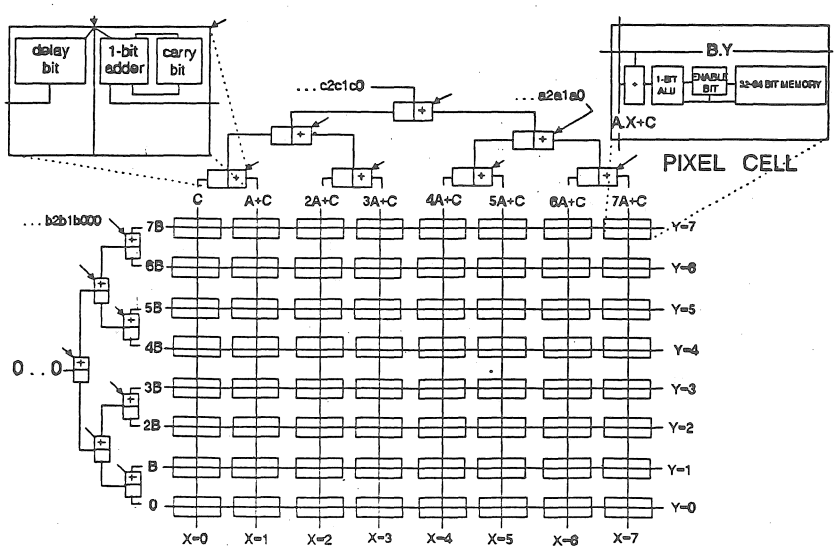


Figura 9 - Arquitetura do Sistema Pixel-Planes 4

A CAD System for Teaching the Design of VLSI Circuits - Status and Evolution -

M. L. Anido and C. E. T. Oliveira

Núcleo de Computação Eletrônica da UFRJ
Cx. Postal 2324 - CEP 20.001 - Rio de Janeiro - Brasil
e-mail:ncd10121@ufrj.bitnet

8 July 1992

Abstract

This paper presents the main characteristics of TEDMOS - a CAD (Computer Aided Design) system for teaching the design of VLSI (Very Large Scale Integrated) circuits. The paper starts by discussing the main assets of TEDMOS, when compared with commercial CAD systems, and presents the high-level characteristics of its main constituent programs. At the end, the paper discusses the evolution of the TEDMOS system.

Keywords: CAD, VLSI, Circuit Simulation, DRC, Extractor, Graphic Editor.

1 Introduction

The development of integrated circuits depends ever more on software tools to, automatically, generate and verify the basic building blocks specified by the circuit designer.

The motivation, sometimes difficult to be understood by researchers in the First World, that led to the development of TEDMOS, can be easily explained by two reasons. Firstly, commercial and powerful CAD packages for VLSI design are very expensive and require powerful workstations. Secondly, these packages have a very long learning time. These reasons forbid these CAD packages to be made available to large groups of students, which is normally the case in undergraduate courses. On the other hand, commercial CAD packages are essential to develop highly complex designs.

TEDMOS is a rather complete CAD package for teaching VLSI design, which is distributed free of charge to universities and research centers. TEDMOS has been highly successful in several Latin American countries among the teaching community, because it runs on IBM-PC© like workstations and because its highly integrated programs increase productivity which compensate other weakness originated from system simplicity. The system is orientated towards full custom design. This design style does not provide the same productivity of standard cell or gate array designs. Nevertheless, it allows very compact designs and also allows students to learn the basic operation of many fundamental transistor structures (cells). Additionally, this approach can be used for digital, analog or mixed designs.

2 An Overview of TEDMOS and its Main Constituent Programs

The TEDMOS system comprises several well integrated programs, that is, the user can invoke some programs within others, without leaving the environment. The main programs of the TEDMOS

system are:

- *File Utility Program* - This program permits to perform several operations on files and directories, without leaving the environment, such as: file deletion, copy, rename, print, change directory, etc.

- *Text Editor* - The text editor allows file editing operations, without leaving the environment, which are sometimes necessary to edit configuration files.

- *Cell Editor* - The cell editor is a graphic editor which allows the creation and modification of cells. This editor is targeted to edit cells and not the complete layout, as opposed to the hierarchical editor described below.

- *Hierarchical Editor* - The hierarchical editor was developed to permit the assembly of the many cells which generally comprise an integrated circuit. It also allows editing individual cells, similarly to the cell editor. The hierarchical editor permits to edit much larger circuits than the cell editor does, because it deals with subtrees of the layout tree and because it uses a more compact data structure than the cell editor does.

- *PLA Generator* - This program accepts logic equations in the form of sum of products and generates the final layout, automatically.

- *Switch-Level Simulator* - Switch-Level simulation can be used before the actual layout of the chip to make sure that the designed circuit implements the planned function. It can also be employed after the layout and extraction phase to verify that the layout implements the planned function. This simulator has a graphic driver capable of displaying the waveforms generated by the simulation.

- *Design Rule Checker* - Each fabrication process is characterized by a set of parameters that describe the valid geometric constructions used in the integrated circuit. This set of rules is called geometric design rules.

- *Extractor* - The circuit extractor takes a geometrical description of the circuit and provides a transistor netlist which can be used by other programs for logical and electrical verification of the layout.

- *Electrical Simulator* - The electrical simulator is composed of three main units, namely: Preprocessor, Electrical Simulator and Graphic Driver. The Pre-Processor converts extracted files to SPICE-like inputs for simulation. The electrical simulator performs a simulation of the circuit, using a Wave Relaxation [5] algorithm , which allows speeding up the simulation. The Graphic Driver makes it possible to visualize the waveform outputs from de electrical simulator, on the screen or on plotters.

3 The Cell Editor of TEDMOS

The graphic editor is the nucleous of TEDMOS [4]. By using it, the designer can create and modify cell layouts of integrated circuits on the computer screen. The screen depicts the complete layout or part of it. The designer moves the rectangular cursor around the screen and orders the editor to fill that rectangle with a certain pattern, which represents one of the fabrication masks (see figure 1). The editor of TEDMOS was designed to operate with orthogonal (Manhattan) geommetry, that is, it is not possible to draw rectangles with inclination.

This cell editor uses an in-memory *bitmap* to represent the data structure of the mask geommetry. Each element of the *bitmap* (matrix) is composed of a set of bits. Each bit indicates the presence or absence of a mask at that point. This representation is only valid for Manhatam or orthogonal geommetries. The algorithms for this type of representation are trivial. However, the memory area required is much larger than it is using alternative data structures. This form of representation is only used internally to the system and it is converted to a more compact form of representation (CIF language) to be archived. The visualization of the editing matrix is performed by mapping each element of this matrix onto a set of screen

pixels.

In order to copy one area of the layout to the screen, it is necessary to encode each element of this area, generating a matrix of points that have to be written onto the screen pixels. This encoding task is accomplished by a look-up table, that is created from an external configuration file. This file is distributed together with the TEDMOS system, but it can be modified by the user.

4 CIRCO - The Hierarchical Editor of TEDMOS

CIRCO is a hierarchical editor, that is, it allows working with a hierarchy of cells which can be represented by blocks that can be interconnected. It was developed to faciliate the assembly of the many cells that usually comprise an integrated circuit [8]. All graphics editing functions of the cell editor can also be found in the hierarchical editor. Additionally, there are special functions for the manipulation of hierarchies. Figure 2 illustrates the interconnection of some cells in CIRCO.

The main groups of commands of CIRCO are:

- *Commands related to Cursor Position and Others* - Among these commands are: Cursor Fill, Cursor Delete, Change Layer, Select Visibility, Change Scaling Factor, Move Window, etc;
- *Area Commands* - The most important commands of this group are: Area Adjust, Area Mirror, Set Position, Area Copy, Return to Position, Repeat Area, Adjust Dimensions;
- *Node Commands* - Among the most important commands are: Create Node, Delete Node, Show/Hide Node, List Nodes;
- *Cell Manipulation and Hierarchy Commands* - The most important commands are: Cell Edit, Cell Store, Create Call to Cell, Remove Call, Move Call, Create Hierarchy, Remove Hierarchy, Expand Hierarchy, Edit Symbol Table.

4.1 The Data Structure of the Hierarchical Editor

Every graphics editing operation involves a Memory Storage Organisation and a Disk Storage Organisation, which are discussed below.

4.1.1 Memory Storage Organisation

Cell editing is an interactive process and in consequence the time required to execute the editing tasks is directly proportional to the time required to access the stored information. Therefore, the data structure has to allow the fast editing of the information.

Sequential search is an important limitation to be considered when defining a data structure, because it is a time consuming operation. In order to solve this problem, a bucket partitioning technique [2] was employed, and it is very efficient for this type of application. The layout is divided into rectangular regions - buckets - of equal size. Each bucket emcompasses a list with the boxes that intercept its region. Figure 3 illustrates the structure of buckets of a hypothetical layout. Because a certain box can occupy an area that intercepts more than one bucket, such box will be associated to several buckets. This limits the sequential search operation to the boxes of a certain area, searching only the buckets that they intercept. The selection of the convenient boxes is done by comparing the minimum and maximum co-ordinates [2]. Care has to be taken to avoid boxes already selected in previous buckets, because a box can occupy an area that incorporates several buckets.

Each box is associated to a layer. This association is represented by several lists of boxes, one for each layer of the layout. The advantages are: (a) speed in the search of boxes for certain layers, because it is not necessary to traverse all the list elements and (b) memory space, eliminating one field in each element of the list. This representation is very efficient to redraw areas by temporal priority of layers. In this case, only the boxes of the visible layers are searched, and only once, for the whole design.

Besides occupying less space in memory, this type

of implementation allows greater speed in the execution of editing operations. These operations are basically:

1. Select the target region;
2. Traverse the data structure, finding the elements which belong to the region;
3. Modify lists associated to the selected elements;
4. Redraw a screen region;

The amount of memory space required to store lists of elements of all cells would be considerably large, once the layout of a complex circuit may incorporate tens of thousands of elements. For this reason, the editing task is performed on one cell at a time. Additionally, editing its subcells is restricted to editing the contour of such subcells. Accordingly, only the elements of the cell being edited have to be stored in memory.

However, the editing task demands that all the hierarchy specified by the user be stored, either for the allocation of a subcell or for the access to its calls. This data structure has to store all the set of cells that constitute the layout and also all the hierarchic structure, created or modified by the user during the editing operation.

A symbol table is used for this purpose and is subdivided into two structures:

- *A Symbol Table* - A table describing the set of cells used.
- *A List of Calls* - Each cell points to a list of its subcells, which can be one of the lists of elements already described above. Only the list of calls of the cell being edited has to be in memory.

According to this organisation, the hierarchical structure is obtained by searching the lists of calls of each subcell. This implementation allows that several parts of the whole layout (each one with an independent hierarchical structure) can be maintained in the set of cells. This allows greater freedom in the design phase of the circuit and during updates of the hierarchical organisation. It is possible to delete a complete subtree of the layout and use it later, once all subcells remain in the symbol table and the structure still remains intact in its lists of calls. The information about each cell, contained in the symbol table is:

- *Number* : Identification used by the program, which is present in the CIF file;
- *Name* : Alternative Identification;
- *Dx,Dy* : Cell Dimensions;
- *File* : File Name Containing its description;
- *Connections* : A description of the connections of each cell.

Part of the symbol table is shown on the screen and it can be edited. The dimensions of each cell were included in the symbol table to be consulted during the drawing operation, if necessary. Therefore, to draw the contour of subcell calls it is not necessary to fetch this information from disk. The same occurs to show the connections of a certain cell, which can also be found in the symbol table. However, information about connections is not shown on the screen, although it is present in the symbol table.

The data structure chosen for the storage of the layout was the implementation of linear lists of elements, with linked lists. Linked lists are more dynamic in insertion and deletion operations, dispensing with preliminary memory space allocation.

4.1.2 Disk Storage Organisation

A disk storage organisation of one file for each cell was found to be appropriated because it is the simplest organisation. This disk organisation allows a sequential file describing its elements for each cell of the circuit, generally in textual format. One of the main aspects considered when choosing a disk layout organisation was the need to provide easy editing of cells coming from non-hierarchical editors, particularly the cell editor of TEDMOS.

In order to fullfill the requirements stated above, the stantard format CIF 2.0 (Caltech Intermediate

Format) was chosen. It allows the description of the design in a portable textual format and it also allows the definition of the hierarchy. Each file defines the elements of a single cell, including the calls to its subcells.

5 The Design Rule Checker

Each fabrication process is characterized by a set of parameters that describes the valid geometric constructions used in the integrated circuit. This set of rules is called geometric design rules.

Due to the extremely large number of transistors and masks in an integrated circuit, it is impossible to verify the correctness of the circuit by visual means. However, this can be done reliably by a computer program that sistematically verifies all the design rules of the circuit.

According to Mead and Conway [6], the geommetric verifications can be classified in categories, such as:

- *Width of Lines* - The dimension of a line of a given layer should be larger than a minimum value.
- *Spacing Rules* - Two geommetric forms of the same layer or of different layers, should keep a minimum distance between them.
- *Extension Rules* - Extension rules are used when a geommetric form should extend over another by a certain value.

The design rule checking task is performed within the editor, that is, the designer does not have to exit the editor to run the DRC. TEDMOS's verification routines are parametric, which means that both the layers and the circuit dimensions involved on each verification routine are read from a file, on execution time. A set of rules is described in this file which specifies rectangle width, rectangle spacing, extensions and transistor construction.

The DRC algorithm employed is a variant of the raster method [3], which is compatible with the form of representation used by the cell editor. The algorithm performs a double scan of the circuit for each rule been scrutinized. This scan operation is performed in two steps: line by line in the horizontal direction and column by column in the vertical direction. The layout is verified as a line of elements, where each element is the set of layers present at that point.

This algorithm is not sufficient to verify all cases and this a deficiency of this DRC program. For example, it does not detect spacing violations at corners. To solve this problem, a new and hierarchical DRC, capable of working with rectangles instead of a *bitmap* description, is under development.

6 The Circuit Extractor

The circuit extractor takes a geometrical description of the circuit and provides a transistor netlist which can be used by other programs for logical and electrical verification of the layout. Basically, the circuit extracting problem consists of the following operations:

1. *Locate transistors, including source, gate, and drain terminals.*
2. *Find transistor connections.*

The circuit extractor of TEDMOS operates on a *bitmap* description of the layout and produces an output file which contains two types of registers: transistors and nodes. The register of transistors associates a number to each transistor and gives a number to each transistor terminal. If two terminals are interconnected by a wire, they will receive the same number. The set of interconnected terminals and wires is termed node. During the editing operation, the designer can give a name to a node to identify simulation points, for example. The registers of nodes identify the name of each node.

This circuit extracting method is based on the *lakes and islands* algorithm [1], which tries to solve the following problem: given a matrix of zeros (0) and ones (1), where 1 indicates ground and 0 water, atribute a code to each contiguous region of ground (island). The solution of this problem is a bidimensional automata that scans the bitmap

from top to bottom and from left to right, attributing a code to each position. This code is a function of the codes of the positions immediately above and immediately to the left. The code zero (0) indicates that the region is water.

This algorithm is capable of extracting wires only. Nevertheless, it can be used to identify peripheral regions of transistors, determining also gates, drains and sources. In order to achieve this, every time that the bitmap contains an intersection of polisilicon with diffusion, this region is marked as a transistor channel. The algorithm is also capable of extracting the capacitances and resistances of wires and transistors, based upon the area of the elements. A new extractor program that takes as input a hierarchical description of the circuit based on rectangles, instead of a *bitmap* description is also under development.

7 The Switch-Level Simulator

A simulator is a program that describes the behaviour of a model of a system. This model can be created with several levels of detail, for example, registers, logic gates and transistors. Each model describes the system in a certain level of abstraction, which can be convenient, depending on the stage of the project. After producing the layout it is very difficult to reconstruct the model at the level of logic gates, and therefore a simulator working at the level of transistors is more adequate.

The switch-level simulator can be used during the design phase to check if a given network of transistors implements the logic operation required. Most important, it can also be used to verify if the implemented layout performs the logic operation desired. This type of verification is fundamental in a full-custom design because of the errors that can occur when editing the layout.

Transistors can be modelled with several degrees of refinement. The more refined the transistor model, the more precise will be the results. However, the computation time will increase. The circuit model used by the TEDMOS switch-level simulator is based on the MOSSIM I simulator [3] and contains two elements: transistors and nodes. Each transistor is modelled as a perfect key, with interchangeable terminals.

Circuit nodes can be in one of the states (0,1,X), where 0 and 1 correspond to low and high voltage levels, respectively, and X denotes an undefined state corresponding to an uninitialized node or a node with a signal value between 0 and 1. Nodes can also be of three types: input (when a node is a voltage power supply with an infinite current supply), Pull-up (when connected to a load transistor) and Normal (none of the previous cases).

A transistor is modelled as a three-terminal (gate, drain, source) bilateral device which could be in any of three states: 0 (open or noconducting), 1 (closed or conducting), and X (undefined, intermediate conductance between its conductance when open and when closed).

The switch-level simulator works well for synchronous circuits or time-independent circuits (e.g. combinational circuits and synchronous finite state machines). The model of transistor used is rudimentary and does not allow the calculation of precise delays or voltage levels. However, this type of simulator provides very fast responses, making it possible to simulate large circuits, which is very useful for a first validation of the circuit.

This is a *unit-delay* simulator, that is, it uses a virtual delay of one time unit for all transistors. The output of the simulator is the output of the steady state of the circuit. It is not possible to analyse the dynamic or transitory behaviour of the transistor network.

A visual interface was created to show the results of the simulation, emulating a ten (10) channel osciloscope and it is shown in the computer screen allowing the simultaneous visualization of 10 points of the circuit. Figure 4 illustrates an example of the output of the switch-level simulator.

8 The Electrical Simulator - ONDAS

The electrical simulator ONDAS uses an input format that is compatible with the SPICE II simulator, except for some restrictions [5]. This for-

mat has the advantage of being very well known among microelectronics designers. Unfortunately, this input format is not user friendly, particularly for the beginner. In order to allow the creation of input files for electrical simulation, a converter program was included in the TEDMOS sytem and it generates a SPICE format, from the output of the circuit extractor. The generated file incorporates a series of informations which are necessary for the electrical simulation and are:

- Technology parameters;
- Dimension of the Elements of the Circuit;
- Capacitance of the Elements of the Circuit;
- Simulation Commands;

ONDAS was developed aiming the reduction of the processing time required by conventional electrical simulators, such as SPICE. Its operation is based on the Wave Relaxation algorithm, that explores some of the characteristics of MOS-VLSI circuits which are stated below:

1. MOS transistors are highly unidirectional devices, that is, source and drain currents are highly dependant of the gate voltage, while the current at the gate node is almost independant of the drain and source voltages.
2. In a MOS-VLSI circuit, typically less than 20% of the node voltages varies significantly in a certain moment of time; ONDAS tries to solve only the equations of active nodes, which represent a significant time reduction.
3. The simulation process occurs in discrete time intervals, termed integration steps. In order to obtain precise results, certain waveform regions, which vary rapidly, have to be calculated in more points, that is, using smaller integration steps.

In conventional simulators, the integration steps are dictated by the worst case, that is, by the waveform that varies more rapidly in the circuit and this sequence of integration steps is used for the whole circuit. The algorithm used by ONDAS allows that one of the waveforms be discretized using its own sequence of integration steps, which implicates considerable reductions of processing time, in comparison with conventional electrical simulators.

The algorithm used by ONDAS has guaranteed convergence only when applied to MOS circuits containing only transistors and capacitors. For this reason the simulator can not be applied to the analisys of circuits containing bipolar transistors, diodes, resistors, inductors and other elements. The convergence of the algorithm used in the simulator is conditioned to the existence of non-zero capacitances connected to each active node of the circuit. By active nodes it should be understood those nodes not connected to the power supplies.

9 PLA Automatic Synthesis

Programmable Logic Arrays (PLA) are structures with regular layout, which are used to implement logic functions. These are mostly used in the control section of integrated circuits. This program accepts logic equations in the form of sum of products and generates the final layout, automatically. A technology file has to be created for each technology.

A PLA consists of two matrices called the planes of the PLA. The most common forms of PLAs are NOR-NOR (two NOR planes) and NAND-NAND (two NAND planes). In order to define a PLA, a language is used to describe the logic equations. For example, a PLA can be described as follows:

```
OP planumber 10;      * cell number
EN A B C;             * normal inputs
SN S1 S2;             * normal outputs
EQ S1 = AB;           * equation
EQ S2 = AB + A'B'C; * equation
```

10 Evolution of the TEDMOS System

The encouragement provided by TEDMOS users, from several countries, has been the driving force to improve the system. New tools to improve the system are under development. They will allow

the design of medium size to large size circuits, using PC-like workstations and other more powerful workstations. Among the improvements and new tools under developemnt are:

- A TEDMOS version for Windows;
- A hierarchical DRC;
- A hierachical Extractor;
- A new version of the Electrical Simulator - ONDAS;
- Porting the system to a SPARC workstation;
- A channel router;
- Automatic synthesis fo regular circuits, such as ROM and RAM.

11 Concluding Remarks

This paper described TEDMOS - a rather complete CAD system for teaching the design of VLSI integrated circuits.

Commercial and powerful CAD software packages for VLSI design are very expensive, require powerful workstations and they also require a continual investment in maintenance and upgrading. Additionally, these packages demand a very long learning time. These reasons prohibit these CAD packages to be made available to large groups of students, which is normally the case in undergraduate courses. TEDMOS fills this gap at zero cost - it is distributed free of charge to universities and research centers.

Presently, TEDMOS runs on IBM-PC© like workstations and its highly integrated programs increase productivity, which compensate other weakness originated from system simplicity.

12 Acknowledgements

The TEDMOS system was originally developed by Dr. Eber A. Schmitz, José A. Borges, Jonas Knopman, Júlio T. C. Silveira, João Assis, Paula Cyrillo and many other students to whom we sincerely thank for their work. We are currently working on the improvements stated on section 10.

References

[1] Baker, C. M. ,"Artwork Analysis Tools for VLSI Circuits", Cambridge, Mass., MIT, 75p, M.Sc. Thesis, (MIT LCS TR-239), 1980.

[2] Borges, J. A. S.,"Editores Gráficos para Projeto de Circuitos Integrados", Rio de Janeiro, COPPE/UFRJ, M.Sc. Thesis, 1987.

[3] Bryant, R. E., "An Algorithm for MOS Logic Simulation", Lambda, Palo Alto, CA, Redwood Systems Group, 1(3): 46-8, 50-3, Fourth Quarter, 1980.

[4] Schmitz, E. A., Borges, J. A. S. and Knopman, J., "TEDMOS - Um sistema de CAD para ensino de projeto de circuitos eletrônicos de alta integração", Revista Brasileira de Computação, Vol. 5, N. 2, pp. 45-61, Out/Dez, 1989.

[5] Knopman, J., Mesquita Filho, A. C., Schechtman, J., "Convergence Properties of Relaxation Methods in the DC Analysis of Large MOS Circuits", Proc. IEEE Int. Symp. on Circuits and Systems. Espoo, Finland, June 7-9, pp. 1639-42, 1988.

[6] Mead, C. and Conway, L. "Introduction to VLSI Systems", Reading, Mass., Addison-Wesley, 396p., 1980.

[7] Schmitz, E. and Borges, J. A. S., "Projeto de Circuitos Integrados CMOS", Rio de Janeiro, LTC, 1989.

[8] Silveira, J. T. C., "CIRCO: Um Editor Gráfico Hierárquico de Circuitos Integrados para Microcomputadores IBM-PC", M.Sc. Thesis, Instituto de Matemática da Universidade Federal do Rio de Janeiro, 61p, 1990.

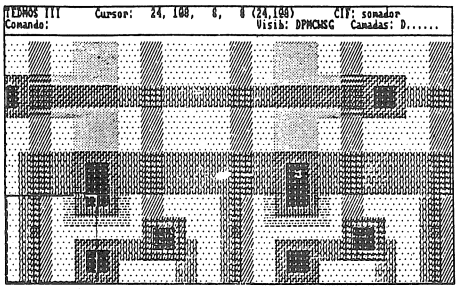


Figure 1 - The Graphic Cell Editor.

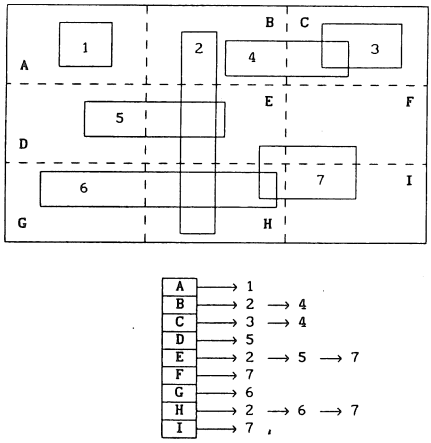


Figure 3 - Partition of The Layout in Buckets

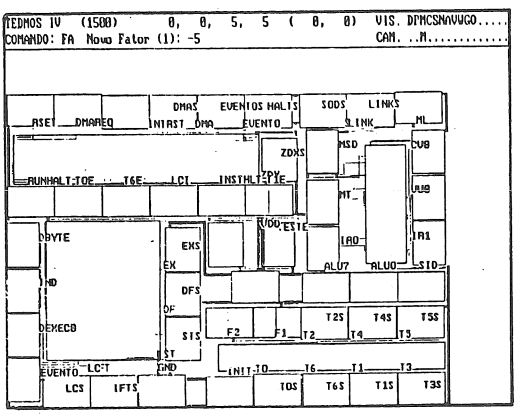


Figure 2 - The Hierarchical Editor.

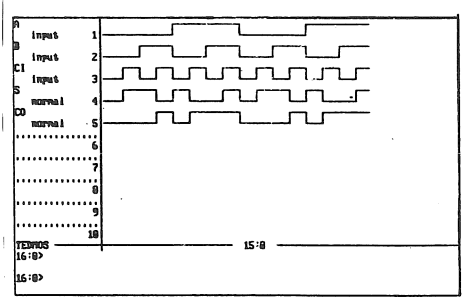


Figure 4 - A Plot Output of the Switch-Level Simulator

ETIQUETADO DE ARTICULOS CIENTIFICOS: un primer intento de construcción de un sistema basado en el conocimiento para tratar ficheros ASCII para publicar con Ventura.

María José Aramburu Cabo[1]
Peter Hammersley[2]

[1]*Universitat Jaume I*
Campus Penyeta Roja
12071 Castellón
Tel.: 964/345775
Fax: 964/345847
España

[2]*Middlesex University*
The Burroughs, Hendon
London NW4 4BT
Inglaterra

RESUMEN

En este artículo presentamos el proceso de desarrollo de un sistema que etiqueta el texto ASCII de un artículo científico con las etiquetas aceptadas por el programa de autoedición Xerox Ventura Publisher. Esta labor, en la actualidad resuelta de forma manual, constituye un eslabón esencial en la automatización de la edición de una revista.

Su mayor novedad se centra en que es una implementación en PROLOG de un reconocedor inteligente de elementos textuales a través de su "comprensión" semántico-sintática de alto nivel. Las reglas utilizadas se basan en la gramática libre de contexto que representa la estructura de la clase de documentos que reconoce.

El sistema resultante trabaja exitosamente para los artículos que guardan la estructura establecida y, además, realiza una ordenación automática de las referencias que se hagan en el texto. Esto permite una edición automática del artículo, que acelera todos los procesos previos a su publicación. Un primer prototipo ha sido aplicado a artículos ASCII recibidos por e-mail en la editorial de la revista científica "The Computer Journal" obteniendose un texto en el que ya no es necesaria la intervención de un maquetador para su publicación.

Como posteriores desarrollos de esta investigación, actualmente se esta trabajando en la linea de crear un sistema basado en el conocimiento que admita más estructuras de documento, nuevas técnicas de marcado a mayor nivel de detalle y que ofrezca un interfaz al usuario que le permita definir el estilo y la forma del texto.

Palabras Clave: Sistema Basado en el Conocimiento, PROLOG, Reconocedor de Texto, Inserción de Etiquetas, Transmisión y Edición de Documentos, SGML, ODA (Office Document Architecture), Gestión de Referencias, Documentos Estructurados.

1. Introducción.

En el mundo académico y de la investigación, la publicación de prensa científica es de vital importancia, ya que supone la comunicación y el esparcimiento de los temas de estudio por todo el área de interés. La publicación de artículos científicos debe ser muy rápida, ya que cuanto antes esté disponible la información para otros científicos, antes serán posibles nuevos desarrollos.

La automatización del proceso de publicar un documento comenzó hace años con el uso de los procesadores de texto, pero la real incursión de la informática en el mundo de las publicaciones llegó con los sistemas de composición controlados por computador. Al mismo tiempo, los lenguajes de marcado de texto también evolucionaron para aprovechar las nuevas posibilidades ofrecidas por la tecnología informática.

Los dos eventos clave para la amplia aplicación actual de la informática a la industria de la publicación han sido [1] el espectacular descenso del costo de los ordenadores y su amplio uso en todos los campos, además de otros hechos como la mejora de los medios de comunicación electrónica y la adaptación de las tecnologías láser para la impresión de documentos con una alta calidad y rapidez.

Los sistemas de autoedición y la publicación electrónica de documentos, son dos nuevos conceptos que representan los últimos desarrollos en este área. Junto con ellos, otros avances en el software como han sido los lenguajes de descripción de páginas, han hecho que la publicación electrónica de documentos y la composición asistida por ordenador, sean unos importantes campos de aplicación de nuevas técnicas informáticas.

1.1 Propósito de este trabajo.

El desarrollo y la publicación de documentos e incluso su acceso por el lector, están evolucionando. Actualmente, el proceso del texto desde que es escrito por el autor hasta que es accesible por el lector, está siendo muy simplificado y acelerado. Esto se refleja en una mayor colaboración entre los participantes en la tarea de publicación.

En el caso concreto de los artículos científicos, se ha intentado asignar al autor la labor de dar forma al documento al mismo tiempo que lo escribe. Para ello, el redactor ha de facilitar al autor un código genérico de composición, lenguaje de marcado o procesador de texto con claras instrucciones y de fácil manejo. Sin embargo, la experiencia ha demostrado los inconvenientes de este método, y la necesidad de buscar nuevas alternativas debido, principalmente, a las dificultades que encuentran los autores para ajustarse a los códigos y procesadores suministrados y su falta de información sobre la apariencia final del texto [2].

Esto hizo pensar en la necesidad de un sistema automatizado que, en lugar del autor, insertara en el texto el código genérico propio del editor; es decir, un programa que diera forma al texto según su estructura específica. Esto evitaría la tediosa labor para el autor de adaptarse a diferentes revistas y eliminaría los errores usuales debidos a su inexperiencia en temas de publicación.

El sistema a desarrollar debía contener todo el conocimiento necesario para determinar la forma que se debe dar a una porción de texto, al tiempo que debía reconocer su estructura (del mismo modo en que lo haría un redactor o editor de documentos).

La solución al anterior reto que se explora en este trabajo, se basa en el desarrollo de un sistema para el caso particular de señalar texto ASCII según el método usado por el programa de autoedición Xerox Ventura Publisher [1]. El sistema resultante basado en técnicas de sistemas basados en el conocimiento, señala cada párrafo de texto con la etiqueta adecuada dependiendo de su posición dentro de la estructura del documento. De esta manera, la composición y el formato del texto podrían ser especificados e imprimidos de manera automática y con el mínimo esfuerzo por parte de autores, redactores y editores.

Además de lo anterior, el sistema ofrece la posibilidad de realizar la ordenación y numeración de las referencias insertadas en el texto automáticamente. Esto libera al autor de tan tediosa labor, y asegura la inexistencia de errores.

La implementación del sistema se realizó en PROLOG mediante la representación en forma de reglas de la gramática libre de contexto correspondiente a la estructura preestablecida para el documento.

En las secciones 2 y 3 describiremos el estado del arte, describiremos en la sección 4 el trabajo realizado, para acabar en la 5 estudiando sus limitaciones.

2. Sistemas de preparación de documentos.

2.1 Marcado de texto.

En el método tradicional de publicación, el marcado de texto consiste en las señales y anotaciones hechas por el diseñador o el redactor con el fin de indicar al compositor cual debe ser la apariencia del documento impreso. En el caso de los procesadores de texto, marcado se refiere a los códigos e instrucciones insertadas en un documento por el programa de que se trate.

Entre los lenguajes y métodos de marcado de texto se distinguen dos tipos:

- **Marcado de procedimiento.** En él se especifica a un bajo nivel de detalle el diseño de cada párrafo. Este método conduce fácilmente a errores, es inflexible a cambios de forma y es de difícil uso para no expertos.
- **Marcado descriptivo.** Se basa en los siguientes postulados que definen sus propiedades [3]:
    - El marcado debe definir la estructura del texto, más que sus características físicas.
    - El marcado debe ser entendido, sin ambigüedad, tanto por programas como por personas.

Con este método, la presentación del texto será determinada por el programa que interprete el marcado. Sus ventajas surgen de su carácter genérico.

Hay que señalar que aunque algunos autores consideran que esta forma de marcar permite separar la estructura de la apariencia del texto, esto no es posible ni deseable [2].

2.2 La evolución de los sistemas de preparación de documentos.

Al mismo tiempo que la tecnología y el costo del hardware y el software han ido evolucionando, los sistemas de preparación de documentos han tratado de balancear el esfuerzo de producción con la calidad del producto.

En los años setenta, los sistemas desarrollados utilizaban un marcado de procedimiento, con el inconveniente de necesitar ser ejecutados para comprobar los efectos especificados en los comandos. Los macroprocesadores permitian definir secuencias de comandos con otros de más fácil manejo (p.ej.: *ms, mm*).

En los años ochenta, se han desarrollado y utilizado extensamente el sistema TEX [4] y el conjunto de programas que lo soportan. Los comandos y parámetros proporcionados por TEX son muchos más que los de *Troff* [5], y además su sintaxis es más regular e incluye nociones para anidar así como de alcance de los comandos.

Una importante derivación de TEX ha sido su macroprocesador LATEX, que establece un modo "lógico" de uso y que, al mismo tiempo, permite un acceso fácil al subyacente TEX. Es necesario señalar, la importante innovación que LATEX supuso por su enfoque lógico del marcado de textos.

El avance más importante (realizado durante los años ochenta) en los sistemas de preparación de documentos, han sido las impresoras láser junto con los **lenguajes de descripción de**

páginas. Este método describe un objeto gráfico como un conjunto de líneas y curvas, e información de su localización en la página. PostScript [6] [7] se ha convertido en un estándar para impresoras láser e incluso para máquinas de composición.

Otro importante avance de los últimos años son los **sistemas de edición interactivos**. En ellos, el usuario interacciona con el sistema y ve los cambios en tiempo real sobre la pantalla (WYSIWYG: What You See Is What You Get). Sin embargo, estos sistemas no son tan potentes como los batch, son más tediosos de usar por los expertos y tienden a concentrarse más en los aspectos visuales que en la estructura lógica del documento [8].

En el futuro, lo deseable sería una combinación de sistemas batch e interactivos. Esto constituye un área de investigación sobre el que se está trabajando actualmente con mucha intensidad.

3. Estándares para documentos estructurados.

Pocos de los métodos actuales para transmisión electrónica de documentos aportan la posibilidad de enviar, junto con el texto, información de cómo el documento será presentado después de impreso[1] o lo que es lo equivalente, su estructura. Esta ausencia de información sobre la estructura del texto, puede causar una degeneración del mensaje transmitido por el autor.

De cara al futuro será necesario un método estándar para definir la estructura de un documento que permita al autor expresar su mensaje sin restricciones y al lector hacer uso de la información recibida de la mejor manera que considere oportuna.

3.1 S.G.M.L.

S.G.M.L. [3][9] (**Standard Generalized Markup Language**) fue diseñado para permitir añadir información estructural a un documento por medio de la inserción de secuencias de caracteres definidas por el usuario en la cadena del texto.

SGML no establece unas reglas de cómo codificar un documento, sino que formaliza un conjunto de reglas que permiten al autor definir cómo ha decidido codificar su documento.

La estructura de un documento SGML, se define formalmente al comienzo del mismo en un conjunto de declaraciones que especifican la clase del documento y que constituyen la **Definición del Tipo de Documento** (DTD). En la DTD se define el **Identificador Genérico** (etiqueta) de cada elemento del documento y el modelo de contenido que define qué subelementos y cadenas de caracteres son aceptables.

El procesamiento posterior depende de la aplicación del documento. Si va a ser impreso, el parser que analice su estructura pasará el control a un programa que sustituya cada etiqueta por el conjunto de comandos del subyacente sistema de composición. En otro caso, el control pasará a otro tipo de procedimiento. Normalmente, cada identificador genérico tendrá asociado un procedimiento distinto.

Entre las muchas ventajas que este lenguaje proporciona al proceso previo a la publicación destacan:

- Facilita el intercambio de documentos.
- El texto puede ser transmitido de forma electrónica y ser impreso con diferentes estilos en los diferentes estados de producción [10].
- Tanto editores como redactores ahorran tiempo y trabajo en el proceso de publicación de documentos.

---

[1] LaTeX es un notable ejemplo de la difusión de un sistema que permite enviar electrónicamente un formato. A pesar de que su manejo puede resultar complejo y sus presentaciones son rígidas, es utilizado por muchos científicos en la actualidad.

- Puede ser usado para preparar documentos por cualquier autor con un procesador de textos. Lo único necesario es utilizar un método de marcado formado por caracteres ASCII.

- Si no se desea crear una DTD por considerarse una difícil labor, el usuario puede utilizar una de las muchas comercializadas actualmente.

- El fichero producido soportará con flexibilidad una gran cantidad de formas de impresión y gestión electrónica y podrá ser usado para diferentes propósitos.

3.2 O.D.A.

ODA [11] [12] (**Office Document Architecture**) como SGML, es un estándar para la representación e intercambio de documentos estructurados.

ODA provee un modelo de documento jerárquico y orientado a objetos. Cada objeto se corresponde con un componente del documento, y los atributos contienen información acerca de los objetos. Un documento es considerado como un árbol, donde la estructura está definida por la forma del árbol y el contenido se almacena enteramente en sus hojas.

ODA asigna dos estructuras al texto dependientes entre sí y que coinciden a nivel de contenido: una estructura lógica que describe las relaciones abstractas entre componentes del texto y una estructura de formato definida en función de páginas, columnas y márgenes.

El estilo de un documento ODA después de impreso, viene determinado por un proceso que da forma al texto. Este proceso se encarga de dividir el contenido en bloques de texto y dependiendo de los atributos de cada objeto, coloca cada bloque en su lugar adecuado en una página.

4. Proceso de etiquetado automático.

4.1 Objetivos y definición del problema.

En las secciones anteriores se han presentado las ventajas y posibilidades ofrecidas por las nuevas tecnologías al proceso de publicar un documento. La principal conclusión de todo ello podría resumirse en que si la comunicación entre los participantes en el proceso de publicación y el tratamiento de la información se hiciera electrónicamente, todo el proceso se mejoraría.

Este trabajo trata de aplicar los resultados anteriores a la publicación automatizada de artículos científicos. Para su desarrollo se toman como base los siguientes puntos [1]:

- La estructura generalmente común de los artículos científicos: título, autor, abstract, cuerpo de texto y referencias.

- La posibilidad de cualquier autor actual de producir documentos en formato ASCII y enviarlos vía e-mail o discos flexibles.

- La actual aceptación de los editores de texto marcado en algún sistema que ellos suministran, y que pueden traducir directamente a código PostScript interpretable por las máquinas de composición.

- La expansión de los programas de autoedición y su aceptación como sistemas de marcado por los editores. En concreto, en este trabajo se adoptó el programa Xerox Ventura Publisher, el cual marca el texto insertando etiquetas delante de cada párrafo.

- La posibilidad de representar con un sistema el conocimiento necesario para reconocer cada una de las partes que constituye un artículo. En el futuro, se espera que los sistemas de publicación por ordenador contengan conocimiento en áreas tales como el estilo de formar una página, la clasificación de documentos y quizá, el diseño de estilos [13].

Con todo ello, el propósito de este trabajo es desarrollar un primer prototipo de sistema basado en el conocimiento capaz de aceptar artículos científicos sin forma definida y contenidos en ficheros ASCII, e insertar automáticamente, las etiquetas adecuadas para Ventura Publisher.

El programa siempre insertará las mismas etiquetas que indicarán la estructura lógica del documento. Las diferencias en el estilo final del texto dependerán de la hoja de estilo que esté definida en Ventura en ese momento. Así, el autor podrá editar su texto como si fuera el propio editor, si este último le suministra la hoja de estilo adecuada (cada editor tiene definida su propia hoja de estilo dependiendo de las características de su publicación).

Es necesario señalar que el prototipo que se quiere desarrollar no se trata de uno de los ya comercializados sistemas de reconocimiento óptico de caracteres (OCR) capaces de reconocer ademas de los caracteres de una página, su formato (fuente, estilo, etc.). No se puede asociar tampoco este sistema a los traductores de formato entre procesadores de texto, puesto que en nuestro caso el documento inicial no se encuentra más que en puro ASCII y sin más referencia a su forma o estilo que su propia estructura.

Cómo resultado se proporcionará un medio de preparación de artículos científicos (transmisión, edición y formato) tan útil para el editor como podría ser el recibirlo ya formateado en LaTeX o SGML pero sin los inconvenientes y restricciones que suponen estos métodos para el autor.

El prototipo reconocedor de texto ha sido implementado en PROLOG principalmente por dos razones:

- Por ser un lenguaje muy adecuado para la elaboración de prototipos, ya que permite al programador abordar el problema a diferentes niveles de abstracción según su conocimiento del mismo.

- Tratarse de un proceso de análisis sintáctico y semántico del texto que requiere una descripción del objeto del problema más que una solución particular.

Como complemento al principal objetivo ya mencionado se ha implementado un tratamiento de las referencias y citaciones del texto que las localiza y numera automáticamente.

4.2 Arquitectura del sistema y metodología de implementación.

El artículo en formato ASCII que va a ser formateado por el sistema debe sufrir varias transformaciones a lo largo de la arquitectura del mismo.

• Eliminación del formato ASCII y Separación de Párrafos.

Este proceso consiste en: eliminar los caracteres de retorno de final de línea y dejar los de final de párrafo y reducir cada secuencia de blancos a uno solo.

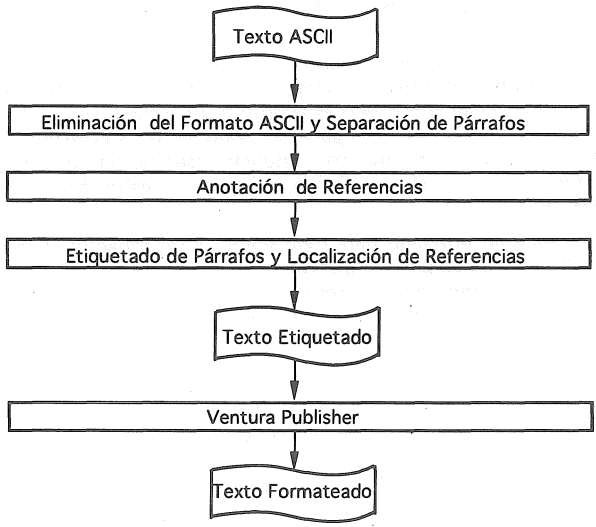
La implementación de esta primera transformación se ha hecho en lenguaje C ya que conlleva un simple análisis léxico. Cada carácter es analizado individualmente, sin tener en cuenta su posición en la estructura del documento. El único conocimiento que se necesita, es la posición del carácter en la línea y cuales son los que le siguen.

• Anotación de referencias.

Después de recorrer todo el texto hasta la sección de referencias, se hace una lista enumerada de todas las referencias que aparecen en la mima.

Esta segunda pasada sobre el artículo ha sido implementada en PROLOG y produce la inserción en la base de conocimiento de PROLOG de un hecho por cada una de las referencias interpretadas con el número que se le asocie.

La posterior conexión entre referencias y citaciones se hace por medio del nombre del autor y su año de publicación. Ambas informaciones han de ser extraidas de todos los datos que conforman una referencia.



- **Etiquetado de Párrafos y Localización de Referencias.**

Como se ha dicho anteriormente, Ventura etiqueta cada párrafo de texto. Por esta razón, definimos los párrafos como elementos básicos.

Este es el proceso clave del sistema y lo componen dos tareas que se ejecutan simultáneamente:

- Verificar la estructura el texto, reconociendo cada una de sus componentes.
- Substituir en el mismo cada una de las citaciones por el número asociado con la referencia correspondiente.

Para verificar la estructura del texto el programa debe localizar la posición del párrafo en la preestablecida estructura del artículo dependiendo del texto previamente recorrido. Su código consiste en una representación interna, por medio de reglas (PROLOG), del posible rango de elementos que pueden formar un documento; es decir, la implementación del programa se basa en el esquema que representa la estructura de un artículo científico, donde cada elemento es un párrafo diferente. En la siguiente sección, se habla de la representación de esta estructura utilizando una gramática libre de contexto y de su transformación en reglas.

El C utilizado para escribir la primera parte del prototipo fue Turbo C sobre un ordenador compatible PC. Posteriormente, este programa fue transmitido al mismo sistema en que fueron escritos los programas en PROLOG. Se trata de un VAX cluster con sistema operativo VMS. El PROLOG utilizado fue el incluido en el entorno POPLOG, desarrollado en la Universidad de Sussex en Inglaterra [14].

4.2.1 Representación gramatical de la estructura.

La especificación lógica del problema, por medio de una gramática libre de contexto, supuso la primera aproximación a la implementación del sistema en PROLOG.

La representación estructural del documento es una visión de alto nivel de su organización lógica. Esta representación puede considerarse a dos diferentes niveles: específico y genérico. La **estructura lógica genérica** representa el modelo para los documentos de una clase (p.ej.: cartas, artículos, manuales, etc). La **estructura lógica específica** representa una instancia en una clase de documento [15].

El sistema basado en el conocimiento desarrollado, es una representación gramatical de las relaciones entre los **objetos atómicos** (párrafos) que definen la estructura de un artículo científico. De esta forma, la estructura lógica específica de un artículo es el árbol que se produce al filtrar su texto. Este árbol forma la base para guiar la traducción de la estructura lógica del artículo a su apariencia física, o lo que es lo mismo, para etiquetar cada párrafo de texto. El razonamiento efectuado por el sistema basado en el conocimiento siempre se realiza hacia adelante, junto con el backtracking estándar del PROLOG.

Por dar una idea del método utilizado para representar la información, supongamos que en la posible estructura definida para un articulo científico, el abstract lo constituyen un conjunto de uno o más párrafos de texto comprendidos entre el título del artículo y su cuerpo principal. Opcionalmente, el abstract podría ir precedido de la palabra 'Abstract'. Una manera de reconocer un párrafo de texto como parte del abstract sería:

> **SI** (Párrafo después de la palabra 'ABSTRACT' **OR** Párrafo después de Título) **AND** Párrafo antes de Cuerpo de Texto
>
> **ENTONCES** Párrafo es Abstract.

4.3 El fichero de entrada.

El fichero de entrada, enviado por el autor, debe cumplir ciertos requerimientos [1]:

- Guardar el formato ASCII y sólo contener caracteres ASCII.
- Contener, al menos, dos retornos al final de cada párrafo.
- Tener la estructura especificada.

Para unos resultados óptimos, el autor debe tener en cuenta los siguientes puntos dependiendo de los resultados que quiera obtener:

l) El texto del fichero etiquetado y su orden, coincide exactamente con el de entrada.

2) Los nombres de todos los autores deben aparecer en un sólo párrafo y sus direcciones van en párrafos independientes y en el mismo orden que los nombres.

3) El abstract debe ir precedido siempre de su título (la palabra "ABSTRACT").

4) La numeración de los títulos de secciones y subsecciones debe llevar un orden decimal lógico. Los títulos sin numerar deben ser de menos de cincuenta caracteres y estar formados de una sola frase. Nunca terminarán en coma, punto y coma o dos puntos.

5) La última sección debe ser la de referencias empezando por su título (la palabra "REFERENCES").

6) Los símbolos que identifican un párrafo de numeración son: '*', '.', '-', '(', '[' (p.ej.: (1), [ii], etc.) y un número o letra seguidos de ')' o '-'. Todos ellos deben ir seguidos de un espacio en blanco. Un número seguido de un punto identifica un título y nunca un párrafo de numeración.

7) Todos los elementos del artículo deben guardar la estructura especificada previamente, hecho que como se puede observar no presenta ninguna restricción, ya que es la usualmente utilizada.

4.3.1 Gráficos, figuras y tablas.

El sistema desarrollado sólo trata con texto, el resto de elementos deben ser provistos por el autor en ficheros separados. Posteriormente, un operador de Ventura creará un marco e insertará en él el fichero con la figura (es el único procedimiento manual que queda en el estado actual del sistema).

La posición exacta del gráfico o figura, debe ser especificada por el autor en un párrafo especial de su texto. Sus dos primeros caracteres serán la secuencia '=>' seguida del nombre del fichero a insertar. El siguiente párrafo de texto es considerado un pie de figura si viene precedido por los símbolos '<='. En ambos casos, se inserta una etiqueta especial para señalar al operador la necesidad de su inserción.

Esta idea puede ser ampliada a textos en los que su forma se relaciona con su contenido (poemas, programas, etc).

4.4 Tratamiento de referencias.

Durante el desarrollo de la investigación, se consideró que el programa dispusiera de una facilidad opcional para manejar las referencias con las siguientes utilidades:

- Reconocimiento de las citas a referencias del texto.
- Inserción del número que el programa asocia con la referencia en el lugar adecuado y entre corchetes.
- Un etiquetado especial para las referencias de la sección correspondiente.

La aplicación trabaja con una sola clase de referencias cuya forma es:

1) Nombre de autor/es con una coma al final.

2) Nombre del libro u otro tipo de documento entre comillas.

3) Cualquier otra información incluyendo en cualquier posición el año de publicación siempre entre paréntesis.

En una referencia, el nombre de cada autor será cualquier combinación de nombres, iniciales y apellidos. Varios autores irán separados por comas y los dos últimos por la palabra 'and' o el símbolo '&'. El autor debe tener en cuenta que el último nombre propio o apellido completo que aparezca en un nombre de autor (no iniciales), será considerado el **nombre de citación**. A continuación se enumeran un conjunto de referencias cuyo nombre de citación podría ser [Smith (1988)]:

Smith, (1988) "Electronic Publishing". Ed. Benjamin. London.

J. Smith, "Electronic Publishing", (1988). Ed. Benjamin. London.

John Smith, "Electronic Publishing". Ed. Benjamin. London (1988).

John J. Smith, (1988) "Electronic Publishing" Ed. Benjamin. London.

Para identificar una cita a referencia en el texto, el autor debe especificar el nombre de citación y el año de publicación entre paréntesis y, ambos, entre corchetes. El programa reconocedor de párrafos, sustituye cada citación por el adecuado número de referencia, al mismo tiempo que recorre el texto. Previamente, el programa debe identificar y numerar todas las posibles referencias de la sección de referencias. Entre otras, es posible reconocer en el texto las siguientes citaciones:

[Smith (1988)]	[Smith et al (1990)]
[Smith & Hole (1989)]	[Smith, Hole and Jones (1991)]

En el caso de dos autores en una referencia, el nombre de citación está compuesto por los dos últimos nombres completos de cada autor con 'and' o '&' entre ellos. Para más de dos autores, el autor del artículo puede escribir en la sección de referencias o en las citaciones todos sus nombres o sólo el de uno de ellos y las palabras et al. En el primer caso, el autor puede opcionalmente poner

todos los nombres de citación o usar el primero de ellos y las palabras et al. En el segundo caso, la citación debe consistir del nombre de citación que aparezca y et al seguido de la fecha. De cualquier forma 'and' y '&' son intercambiables.

Si por algún error, una citación no se corresponde con ninguna referencia, el sistema no modifica el texto pero da un mensaje de error al usuario.

El etiquetado de la sección de referencias se diferencia de los demás en que las comillas que rodean al título son sustituidas por los códigos de Ventura que hacen aparecer el texto en cursiva.

La principal ventaja de utilizar la numeración automática de referencias, es que el autor puede insertar, borrar o modificar con la seguridad de que al final no habrá errores en la numeración. Las referencias, dentro de su sección, pueden estar ordenadas al gusto del autor. Su numeración es hecha por el sistema en el mismo orden en que aparecen.

5. Limitaciones del sistema.

El sistema desarrollado, entre otras cosas, trata de ser un trabajo en el que se muestren las grandes posibilidades que los métodos de inteligencia artificial ofrecen para solucionar problemas en el área presentada. En el limitado tiempo asignado para su desarrollo, se trató de ofrecer un mayor número de posibilidades, a cambio de restarle operatividad al sistema; como consecuencia, el sistema presenta algunas limitaciones [1]. Entre ellas:

- El ASCII que reconoce está reducido al conjunto básico de caracteres (ASCII del entorno POPLOG). Sin embargo, cualquier código ASCII entre los símbolos '< >' será traducido por Ventura en su correspondiente carácter.

- Dado que algunos procesadores de texto traducen los tabuladores a espacios, el sistema no los acepta ya que los reduce a un sólo espacio.

- Al igual que Ventura, el sistema no acepta notas a pie de página.

- Los errores en la estructura del texto o en la enumeración de títulos, figuras o párrafos no son tratados.

- Los métodos de numeración y los caracteres especiales para marcado de párrafos son limitados.

- El sistema debería haber sido implementado para su funcionamiento sobre un PC-compatible (la versión de Ventura Publisher que se utilizó era para PC). Sin embargo, por tratarse de un prototipo esto se ha sacrificado en virtud del mejor aprovechamiento de los recursos disponibles.

Como puede observarse, algunas de estas limitaciones pueden ser fácilmente eliminadas y otras necesitan una mayor investigación que podría ser tema de un futuro trabajo.

6. Conclusiones.

El prototipo resultante de esta investigación trabaja exitosamente para los artículos que se ajustan a la estructura especificada por el sistema y que cumplen ciertas reglas, actualmente empleadas y por tanto no restrictivas, en el proceso de mecanografiado en el ordenador.

El sistema en las últimas etapas de su desarrollo se ha aplicado a diversos textos ASCII en Middlesex University de Londres, para el formato de presentación de la revista científica *"The Computer Journal"* que allí se edita. Los resultados obtenidos coinciden con los maquetados manuales, ofreciendo el sistema además de rapidez y seguridad, la opción de tratamiento de referencias, que se ha mostrado muy útil.

En futuros trabajos se considerará el abordar muchas más estructuras de documentos, así como posibilitar más técnicas de etiquetado. El resultado será un formateador inteligente de texto que

permita editar documentos mecanografiados en diferentes procesadores de texto y, posteriormente, publicarlos en una amplia gama de formas definidas por el usuario, disponiendo para ello de varios lenguajes de marcado.

7. Agradecimientos.

Este trabajo ha sido llevado a cabo en Middlesex University de Londres gracias a la concesión por la Universidad Politécnica de Valencia de una beca ERASMUS.

Así pues, deseo agradecer a la Universidad Politécnica de Valencia y al proyecto europeo para el intercambio de estudiantes (ERASMUS), la oportunidad que nos dieron de desarrollar este trabajo, y a Middlesex University por su buena acogida y medios prestados.

También quiero dar las gracias a mis compañeros en Middlesex University y a mi director y colaborador Peter Hammersley, por su confianza y apoyo. Por último, doy gracias a Francisco Toledo de la Universitat Jaume I de Castellón por su ayuda y supervisión al redactar este artículo.

Referencias.

[1] Aramburu, M. J. (1991): *Tagging of Scientific Papers: An Expert System to Prepare ASCII Text for Publication by Ventura*: Middlesex Polytechnic, London.

[2] Hammersley, P. (1985): "The Use of Generic Coding in the Produccion of Journals and Conference Proceedings", *International SGML Conference*, Manchester.

[3] Goldfarb, C. F. (1990): *The SGML Handbook*: Ed. Claredon Press, Oxford.

[4] Donald, A y Knuth, E. (1984): *The TEXbook*: Ed. Addison-Wesley Publishing Company, USA.

[5] Ossanna, J. (1982): *Nroff & Troff. Users Manual and Addendum to the Troff Users Manual*: Bell Laboratory, USA.

[6] Holzgang. (1989): *PostScript Programmers Reference Guide*: Ed. Scott, Foresman and Company, USA.

[7] Reid, C. G. (1988): *PostScript Language. Program Design Adobe Systems Incorporated*: Ed. Addison- Wesley Publishing Company, USA.

[8] Lamport, L. (1988): *Document Production: Visual or Logical?*: Ed. TUGboat.

[9] Bryan, M. (1988): *SGML an Authors Guide to the Standard Ceneralized Markup Language*: Ed. Addison- Wesley Publishing Company, UK.

[10] Smith, J. M. (1986): "The Implications of SGML for the Preparation of Scientific Publications", *The Computer Journal*, Vol. 29, Nº 3: Ed. Cambridge University Press, UK, 193-200.

[11] Brown, H. (1989): "Standars for Structured Documents", *The Computer Journal*, Vol. 32, NQ 6: Ed. Cambridge University Press, UK, 505-514.

[12] Guardalben, G y Giacomello, M. (1990): "An ODA Page Planner for Professional Publishing EP90", *Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography*: Ed. Cambridge University Press, USA.

[13] Kist, J. (1987): *Electronic Publishlng. Looking for a Blueprint*: Ed. Croom Helm, UK.

[14] Systems Designers (1986): *POPLOG User Guide*: Ed. University of Essex, UK.

[15] Furuta, R., Quint, V. y André, J. (1988): "Interactively Editing Structured Documents", *Electronic Publishing-odd*, Vol. 1(1): Ed. John Wiley & Sons, UK, 19-44.

A family of cuts for the matroidal knapsack problem

Julián Aráoz
Universidad Simón Bolívar
Caracas, Venezuela

Francesco Maffioli
Politecnico de Milano
Milano, Italy

Abstract

A family of cuts is given for the polyhedron of the intersection of a matroid with a knapsack problem. These cuts could be combined with matroid and knapsack cuts in a Branch and Cut Method, for this we give some heuristics to obtain such cuts. Some of the cuts give facets of small problems that can be lifted to facets of large problems.

Keywords: Combinatorial Optimization, Polyhedra, Matroids, Knapsacks, Facets

1 Problem statement and definitions

We shall be concerned about Independent Systems, one corresponding to the *Knapsack Problem*, where we assume that each element j of E is given a non-negative integer weight a_j and that the family of independent sets, denoted by $\mathcal{K}$, is defined as the family of subsets I of E such that $\sum_{j \in I} a_j \leq b$, where b is a given positive integer.

The other special Independent System we shall be considering is obtained when the following property holds for $\mathcal{I}$: for all $I, J \in \mathcal{I}$ such that $|I| = |J| + 1$, there exists an element k of $I \setminus J$ such that $I \cup \{k\} \in \mathcal{I}$. In this case the Independent System is called a *Matroïd* and its family of independent sets is denoted by $\mathcal{M}$. The reader is referred to [5] for Matroid Theory fundamentals and notation.

This work studies properties of the Independent System resulting of the intersection of matroid and knapsack Independent System defined over the same ground set E. The simplest non-trivial such intersections is obtained when the family of independent sets of an Independent Systems must obey both a knapsack and a matroid constrains. We then speak of *Matroid Knapsack* problems and we denote by $\mathcal{L}$ the family of its independent sets. These problems have been considered in [2] from the point of view of lagrangean relaxation and its properties. The case of several knapsack constraints has been studied in [1]. Here we begin to analyze the polyhedral structure of these problems.

The polyhedral approach identifies an Independent System using the characteristic vector x^I given for each I by $x^I_j = 1$ if j in I, 0 otherwise, and considering the convex hull of these vectors, i. e. $P_{\mathcal{I}} := conv\{x^I \in \{0,1\}^n : I \in \mathcal{I}\}$.

We denote by $\mathcal{D}$ the family of all subsets of E which are not independent, called the family of *dependent sets* $\mathcal{D} = \{D \in E : D \notin \mathcal{I}\}$, and by $\mathcal{C}$ the family of all minimal dependent sets or *circuits* of the Independent System.

For any circuit C of Independent System a valid inequality for the corresponding polyhedron $P_{\mathcal{I}}$ is given by $\sum_{j \in C} x_j \leq |C| - 1$

In order to simplify the notation in the inequalities, we use $x(A)$ as a short-hand for $\sum_{j \in A} x_j$, for any subset A of E.

More generally, for any $A \subseteq E$, a valid inequality for $P_{\mathcal{I}}$ is given by

$$x(A) \leq r(A) \tag{1}$$

where $r(A) = \max\{|I| : I \in \mathcal{I} \text{ and } I \subseteq A\}$, is called the rank function of the Independent System.

It is well known that $P_{\mathcal{M}}$ has been fully characterized in [3] as the set of real vectors x satisfying the inequalities (1) and non-negativity. As for as $P_{\mathcal{K}}$ is concerned, no full characterization is known, but this polyhedron has been extensively studied and several families of facet-inducing inequalities have been found. The reader is referred to [4] as an

up-to-date text in integer programming and combinatorial optimization.

2 Valid inequalities for Matroidal Knapsack problems

We will derive cuts for $P_{\mathcal{L}}$ from $P_{\mathcal{K}} \cap P_{\mathcal{M}}$.

2.1 Trough the paper, we denote by (P, Q, S, T) a partition of E, with cardinalities p, q, s, t respectively, by k, the rank of $(P \cup T)$ in the matroid $(E, \mathcal{M})$ and by q_0 a distinguish element of Q.

Assumption 2.2 We assume: (a) $t \geq 2$, $q \geq 1$, $0 \leq p < k < p+t$. (b) $r_{\mathcal{M}}(Q \cup P \cup T) = k+q$. (c) The set $P \cup T$, which is dependent in the matroid $(E, \mathcal{M})$, is independent in the knapsack $(E, \mathcal{K})$. (d) For all $e \in T$ the set $P \cup Q \cup \{e\}$ is a cycle of the knapsack $(E, \mathcal{K})$ and is independent in the matroid $(E, \mathcal{M})$. (e) The set S, which represent the elements not used in the configuration, could be empty.

We do not know a complete description for $P_{\mathcal{K}}$, hence we do not expect to find a complete description for $P_{\mathcal{L}}$. But it has to satisfy the rank inequalities (1) for both $P_{\mathcal{K}}$ and $P_{\mathcal{M}}$ and any non-negative linear combination of them. In the next lemma we identify some valid inequalities for $P_{\mathcal{L}}$ corresponding to Assumption 2.2 that we use to derive the cuts.

Lemma 2.3 Apart from $0 \leq x_e \leq 1 \; \forall e \in E$, the following inequalities are valid for $P_{\mathcal{L}}$.

$$x(P \cup T) \leq k \tag{2}$$

$$x(P \cup Q \cup \{e\}) \leq p + q \; \forall e \in T \tag{3}$$

$$2x(T) + (t+1)x(P) + tx(Q) \leq k + t(p+q) \tag{4}$$

$$2x(T) + (t+1)x(P \cup Q) \leq k + t(p+q) + q \tag{5}$$

$$2x(T) + (t+1)x(P \cup Q \setminus \{q_0\}) + tx_{q_0} \leq k + t(p+q) + q - 1 \tag{6}$$

$$2x(T) + (t+2)x(P) + tx(Q) \leq k + t(p+q) + p \tag{7}$$

Proof: The inequalities (2) and (3) are a direct consequence of inequality (1) and Assumption 2.2. Adding the inequality in (2) and the inequalities in (3) we obtain (4). Adding $x_e \leq 1$ for all $e \in Q$ to (4) we obtain (5). Adding $x_e \leq 1$ for all $e \in Q \setminus \{q_0\}$ to (4) we obtain (6). Adding $x_e \leq 1$ for all $e \in P$ to (4) we obtain (7). □

In the next lemma we define x^o, a fractional vertex of $P_{\mathcal{M}} \cap P_{\mathcal{K}}$ which is cut by the inequalities of Theorem 2.5.

Lemma 2.4 Let x^o be defined as: $x^o_e = 1$ if $e \in P \cup Q \setminus \{q_0\}$; $x^o_e = \frac{k-p}{t}$ if $e \in T$; $x^o_e = 1 - \frac{k-p}{t}$ if $e = q_0$; $x^o_e = 0$ if $e \in S$.

Then x^o is a *fractional vertex* of $P_{\mathcal{M}} \cap P_{\mathcal{K}}$.

Proof:

1) Clearly $\vec{0} \leq x^o \leq \vec{1}$ and $0 < x^o_{q_0} < 1$ since $p < k < p + t$ by Assumption 2.2, hence it is fractional. Therefore we could write:

$$x^o = x^{P \cup Q \setminus \{q_0\}} + (1 - \tfrac{k-p}{t}) x^{\{q_0\}} + (\tfrac{k-p}{t}) x^T \quad (a)$$

$$\leq x^{P \cup Q \setminus \{q_0\}} + (1 - \tfrac{k-p}{t}) x^{\{q_0\}} + (\tfrac{t-1}{t}) x^T \quad (b)$$

2) Since (a) is equivalent to:

$$(1 - \frac{k-p}{t}) x^{P \cup Q} + (\frac{k-p}{t}) x^{P \cup T \cup Q \setminus \{q_0\}}$$

We have $x^o \in P_{\mathcal{K}}$, because x^o is a convex combination of two points in $P_{\mathcal{K}}$.

3) Since (b) is equivalent to:

$$(1 - \tfrac{k-p}{t}) x^{P \cup Q} + (\tfrac{t-1}{t})(1 - \tfrac{k-p}{t}) x^T + (\tfrac{k-p}{t}) x^{P \cup Q \setminus \{q_0\}} + (t-1)(\tfrac{k-p}{t^2}) x^T$$

$$= \textstyle\sum_{e \in T} \frac{1}{t}(1 - \frac{k-p}{t}) x^{P \cup Q \cup T \setminus \{e\}} + \sum_{e \in T} \frac{k-p}{t^2} x^{P \cup Q \setminus \{q_0\} \cup T \setminus \{e\}}$$

We have $x^o \in P_{\mathcal{M}}$, because x^o is lest than or equal to a convex combination of points in $P_{\mathcal{M}}$.

4) $x^o(P \cup T) = x^o(P) + x^o(T) = p + (k - p) = k$

$x^o(P \cup Q \cup \{e\}) = p + (q - 1) + (1 - \frac{k-p}{t}) + (\frac{k-p}{t}) = p + q \ \forall e \in T$

Hence the equation system of x^o contains:

$$\begin{aligned}
&\text{i)} \quad x(P \cup T) = k \\
&\text{ii)} \quad x(P \cup Q \cup \{e\}) = p + q \quad \forall e \in T \\
&\text{iii)} \quad x_e = 1 \quad \forall e \in P \cup Q \setminus \{q_0\} \\
&\text{iv)} \quad x_e = 0 \quad \forall e \in S
\end{aligned}$$

The only solution to system (i,ii,iii,iv) is x^o since deducting $x_e = 1 \ \forall e \in P$ from (i), and $x_e \overset{\circ}{=} 1 \ \forall e \in Q \setminus \{q_0\}$ from (iii) we obtain:

i') $x(T) = k - p$

ii') $x_e + x_{q_0} = 1 \ \forall e \in T$

Therefore adding (ii') for all $e \in T$ and using (i'), we have

$x(T) = t(1 - x_{q_0}) = k - p.$

That is $x_{q_0} = 1 - \frac{k-p}{t}$ and $x_e = \frac{k-p}{t} \ \forall e \in T$. Hence x^o is a basic solution.

□

The main result of this paper is the next theorem.

Theorem 2.5 The following inequalities are cuts for $P_{\mathcal{L}}$.

$$x(T) + tx(P) + (t-1)x(Q) \leq t(p+q) - q \tag{8}$$

$$x(T \cup Q) + 2x(P) \leq k + p + q - 1 \tag{9}$$

When t and $k + t(p+q) + q$ are odd:

$$x(T) + \frac{t+1}{2} x(P \cup Q) \leq \frac{k + t(p+q) + q - 1}{2} \tag{10}$$

When t is odd and $k + t(p+q) + q$ is even:

$$x(T) + \frac{t+1}{2} x(P \cup Q \setminus \{q_0\}) + \frac{t-1}{2} x_{q_0} \leq \frac{k + t(p+q) + q - 2}{2} \tag{11}$$

When t is even and $k + t(p+q) + p$ is odd:

$$x(T) + \frac{t+2}{2}x(P) + \frac{t}{2}x(Q) \leq \frac{k + t(p+q) + p - 1}{2} \tag{12}$$

When t and $k + t(p+q) + p$ are even:

$$x(T) + \frac{t+2}{2} + x(P) + \frac{t}{2}x(Q) \leq \frac{k + t(p+q) + p - 2}{2} \tag{13}$$

Proof : The proofs of cuts (8) and (9) are similar. We prove only (8).

Replacing x^o in (8) we have:

$x^o(T) + tx^o(P) + (t-1)x^o(Q) =$

$(k-p) + tp + (t-1)(q-1) + (t-1)(1 - \frac{k-p}{t}) =$

$(k + t(p+q) - q) + \frac{k-p}{t} > k + t(p+q) - q$

To show that it is a valid inequality for $P_{\mathcal{L}}$, consider any set $A \in \mathcal{L}$.

If $P \not\subseteq A$ then $x^A(P) \leq p - 1$ and the vector x^A satisfy (8) since

$x^A(T) + tx^A(P) + (t-1)x^A(Q) \leq t + t(p-1) + (t-1)q = t(p+q) - q.$

If $P \subseteq A$ then:

either $Q \subseteq A$ which implies that not element of T is in the solution, in this case the vector x^A satisfy (8) since

$x^A(T) + tx^A(P) + (t-1)x^A(Q) = 0 + tp + (t-1)q = t(p+q) - q.$

or $x^A(Q) \leq q - 1$ and $x^A(T) \leq k - p \leq t - 1$, because $k = r(P \cap T)$ and $k < t + p$, in this case the vector x^A satisfy (8) since

$x^A(T) + tx^A(P) + (t-1)x^A(Q) \leq t - 1 + tp + (t-1)(q-1) = t(p+q) - q.$

The proofs of cuts (10), (11) and (12) are similar, but with a different technique from the proof of (8) and (9). We prove only (10).

If t and $k + t(p+q) + q$ are odd then the inequality (10) is a valid inequality for $P_{\mathcal{L}}$ which cuts x^o.

Dividing by two the valid inequality (5)

$2x(T) + (t+1)x(P \cup Q) \leq k + t(p+q) + q$

and taking the floor of it right-hand-side gives (10).

Replacing x^o in twice (10) we have:

$2x^o(T) + (t+1)x^o(P \cup Q) =$

$2(k-p) + (t+1)(p+q-1) + (t+1)(1 - \frac{k-p}{t}) =$

$2k - 2p + tp + tq - t + p + q - 1 + t + 1 - k + p - \frac{k-p}{t} =$

$k + (-2 + t + 1 + 1)p + (t+1)q - \frac{k-p}{t} =$

$(k + t(p+q) + q - 1) + (1 - \frac{k-p}{t}) > k + t(p+q) + q - 1.$

The proof of cut (13) uses cuts (8) and (9). □

3 Facets for $P_{\mathcal{L}}$

In this section we give an example that shows how to derive conditions which assure us that some of the inequalities in Theorem 2.5 correspond to facets of $P^*_{\mathcal{L}} = P_{\mathcal{L}} \cap \{x_e = 0 : e \in S\}$.

Hence, we can use standard Lifting Techniques to obtain facets of $P_{\mathcal{L}}$.

To show that a given valid inequality of a polyhedron P is facet inducing it is enough to show that that there are dimension of P points in P which are linear independent and satisfy the given inequality as equality. Notice that dimension of $P^*_{\mathcal{L}}$ is $p+q+t$.

Example 3.1 Let $E = \{1, \ldots, t, p_0, q_0\}$, $T = \{1, \ldots, t\}$, $P = \{p_0\}$ and $Q = \{q_0\}$, with $p = q = 1$ and $k = t$ and let $T \cup Q \in \mathcal{K}$. In this case the dimension of $P^*_{\mathcal{L}}$ is $t+2$.

In $P^*_{\mathcal{L}}$, the $t+2$ points $x^{P \cup T \setminus \{e\}}$, $\forall e \in T$, $x^{P \cup Q}$, $x^{T \cup Q}$ are independent since it is easy to solve. The inequality (8) is facet inducing for $P^*_{\mathcal{L}}$ since in this case this inequality is: $x(T) + tx(P) + (t-1)x(Q) \leq 2t - 1$ which is satisfied as equality by the given points.

4 Separation heuristics for $P_{\mathcal{L}}$

In this section we consider some examples of separation heuristics for $P_{\mathcal{L}}$ than can be used in a Branch and Cut Method.

Given a solution x to the linear relaxation, by Lemma 2.4 we are looking for a dependent set in the matroid with values α and 1 in x, the 1's correspond to P and the others to T. Then look for an arc, independent from the cycle, with value $1-\alpha$ corresponding to q_0 and check if one of the corresponding inequalities is violated, if not, try to add elements to T, P and Q.

We define the submatroid M_α in the ground set $A = \{a \in E : x(a) = 1 \vee x(a) = \alpha \pm \epsilon\}$, $x(a)$ is the weight of a.

For each α present we could look in M_α either for small configurations, which we could enumerate or for a maximum base, this gives a cycles basis and we have a configuration for each element not in the basis or for a maximum 1-forest, that is a forest with one cycle.

References

[1] P. M. Camerini, F. Maffioli and C. Vercellis. "Multiconstrained Matroidal Knapsack Problems", *Mathematical Programming*, Vol. 45, pp 211–231, 1989.

[2] P. M. Camerini and C. Vercellis. "The Matroidal Knapsack: A class of (often) Well-Solvable Problems", *Operations Research Letters*, Vol. 3, pp 157–162, 1984.

[3] E. Edmonds. "Submodular Functions, Matroids and Certain Polyhedral", *Proceedings of the International Conference in Combinatorial Structures and Their Applications* (R. Guy et al. edts.), Gordon and Breach, New York, pp 69–87, 1970.

[4] G. Nemhauser and L. Wolsey. *Integer Programming and Combinatorial Optimization*, North-Holland, Amsterdam, Holland, 1988.

[5] D. J. A. Welsh. *Matroid Theory*. Academic Press, London, 1976.

Un Modelo de Hoja de Cálculo: Observaciones a Implementaciones

Emely Arráiz, Roger Soler, Cristina Zoltan

Universidad Simón Bolívar,Apartado 89000, Caracas , VENEZUELA

Resumen

Se presenta la especificación semántica del lenguaje subyacente en una familia de aplicaciones conocidas como "Hoja de Cálculo", con el propósito de realizar un análisis de esta familia.

Palabras Clave: Hoja de Cálculo, Especificación denotacional,Análisis de programas

1 Introducción

La Semántica Denotacional, algunas veces denominada Semántica Matemática, otras Semántica Funcional es un método desarrollado con el objetivo de describir los conceptos que aparecen en los lenguajes de programación. Este método consiste en asociar a cada constructor sintáctico del lenguaje de programación, a través de funciones de evaluación semántica, valores abstractos u objetos matemáticos. Esta asociación está dirigida por la morfología del lenguaje de programación. La validez del método está fundamentada en lo que se conoce como Teoría de Dominios, [3],[1],[2]. El propósito de este trabajo es usar el método, como lenguaje de especificación, en el proceso de análisis y prueba de aplicaciones.

2 El problema

Una Hoja de Cálculo puede ser vista como un sistema de definición de funciones:

$$y_i \quad = E_i \tag{1}$$

donde los E_i son expresiones con variables acotadas y bien construidas en algún álgebra y y_i son los identificadores de las expresiones (funciones). Por ejemplo:

$$\begin{cases} y_1 & = x_1 * x_2 + x_3 \\ y_2 & = x_3 + 7 + x_1 \\ y_3 & = 121 \end{cases}$$

en este caso podemos estar en el álgebra de los enteros $\mathcal{Z}$, con las operaciones de multiplicación (*) y suma (+) , 7 y 121 constantes y las $x_i, i = 1, 2, 3$ representan variables acotadas. Para los y_i se tiene que son funciones definidas en $y_i : \mathcal{Z}^n \rightarrow \mathcal{Z}$. Se puede presentar (1) como:

$$z_i = y_i(w_i) \tag{2}$$

Continuando el ejemplo anterior, tenemos que los w_i son identificadores de tripletas de valores en $\mathcal{Z}^3$ y los z_i son identificadores de valores en $\mathcal{Z}$ (cada z_i estará asociado al valor de la evaluación de la función y_i en w_i). Si tomamos $w_1 = w_2 = w_3 =< 3, 4, 5 >$ entonces, (2) se reescribirá como:

$$\begin{cases} z_1 = y_1(< 3, 4, 5 >) = 3 * 4 + 5 = 17 \\ z_2 = y_2(< 3, 4, 5 >) = 5 + 7 + 3 = 15 \\ z_3 = y_3(< 3, 4, 5 >) = 121 \end{cases}$$

En el caso de la Hoja de Cálculo, los identificadores de funciones y_i, los identificadores de los argumentos w_i y los identificadores de los resultados de evaluación z_i son "sintácticamente" los mismos que los identificadores de las variables acotadas x_i . Es decir tenemos:

$$\begin{aligned} w_1 &= w_2 = \ldots = w_n = w \\ z &\equiv y \equiv w \equiv x \end{aligned} \tag{3}$$

Usando (3), la evaluación del sistema de funciones presentadas en (1) tiene la siguiente forma:

$$x_i = x_i(x_1, x_2, \ldots, x_n) \tag{4}$$

De esta manera cada identificador x_i tiene asociado una función y un valor que es el resultado de evaluar la función que tiene asociada; además las variables acotadas en las funciones son las mismas x_i.

El sistema de funciones corresponde a las variables de la Hoja de Cálculo y el sistema hoja de cálculo estará constituido por un ambiente y una presentación.

El ambiente fija el tipo de cada uno de los identificadores de la hoja y la secuencia de evaluación del sistema constituido por estas.

El sistema planteado dará resultados diferentes dependiendo de la secuencia de evaluación del mismo, por ejemplo:

- para la secuencia de cálculo: x_3, x_2, x_1 , $x =< 3, 4, 5 >$; se obtiene $x =< 512, 131, 121 >$.

- para la secuencia de cálculo: en paralelo $x_1 \parallel x_2 \parallel x_3$, $x = < 3,4,5 >$; se obtiene $x = < 17,15,121 >$.

Como los identificadores están asociados a valores y funciones, podemos expresar las funciones con argumentos implícitos o explícitos, obteniendose tres tipos de asociaciones para los identificadores. Si agregamos al sistema del ejemplo esta cuarta definición:

$$x_4 = x_1 + x_1() + x_1(x_1 : 6, x_2 : 7, x_3 : 8) \tag{5}$$

tenemos, para la secuencia de cálculo x_1, x_2, x_3, x_4 y $x = < 3,4,5,0 >$ se obtiene $x = < 17,15,121,392 >$.

3 Descripción Denotacional del Lenguaje

A continuación se propone un lenguaje para soportar algunas de las propiedades mencionadas en el párrafo anterior. Primero se presenta una descripción de la parte sintáctica del lenguaje (3.1) y luego una descripción de la semántica (3.2 y 3.3).

3.1 Dominios Sintácticos

El lenguaje está compuesto por las clases sintácticas siguientes:

- Regiones $\mathcal{REG}$ ($\Upsilon \in \mathcal{REG}$) ($\Delta \in \mathcal{VAR} \subseteq \mathcal{REG}$)
- Ordenamientos $\mathcal{ORD}$ ($\Theta \in \mathcal{ORD}$)
- Expresiones $\mathcal{EXP}$ ($E \in \mathcal{EXP}$)
- Comandos $\mathcal{CMD}$ ($\Gamma \in \mathcal{CMD}$)

Los identificadores Υ,Δ, Θ , E y Γ representan elementos genéricos de los dominios de regiones,variables, ordenamientos, expresiones, y comandos respectivamente.

3.1.1 Dominio de Regiones ($\Upsilon \in \mathcal{REG}$)

Las regiones son subconjuntos de variables, y los elementos de este dominio (Υ) son identificadores que incluyen a los identificadores de variables (Δ) que son regiones unitarias. Los nombres de variables están pre-determinados y son infinitas. El dominio $\mathcal{VAR}$ posee un predicado de igualdad que establece cuando dos identificadores de variables son sintácticamente iguales. Este dominio tiene una organización bidimensional, es decir $\mathcal{VAR} = \mathcal{VAR}_1 \times \mathcal{VAR}_2$ y existe un orden total $(\mathcal{VAR}_1, \leq_1), (\mathcal{VAR}_2, \leq_2)$ y operaciones de sucesor y predecesor sobre $\mathcal{VAR}_1$ *y* $\mathcal{VAR}_2$

La morfología de los identificadores de variables está dada por:

$\Delta \longrightarrow < \delta_c, \delta_f >$ denotando columna fila

δ_c es el identificador de las columnas y δ_f el identificador de las filas. En el texto usaremos Δ ó $< \delta_f, \delta_c >$ cuando se requiera. La morfología de los identificadores de región no-unitaria es irrelevante.

3.1.2 Dominio de Ordenamientos ($\Theta \in \mathcal{ORD}$)

En este dominio los identificadores corresponden a las secuencias de evaluación que se utilizarán durante la ejecución de la hoja de cálculo. Las secuencias que se considerarán serán solamente las secuencias finitas ya que corresponden a computaciones finitas.

Al igual que los identificadores de región, su morfología es irrelevante. Sin embargo tendremos elementos distinguidos correspondientes a evaluaciones usuales. Podemos destacar Θ_0 como el identificador de la secuencia actual durante una evaluación; Θ_1 como el identificador de la secuencia de dependencia funcional; Θ_2 como el identificador de la secuencia considerando las variables por filas; Θ_3 como el identificador de la secuencia considerando las variables por columnas; Θ_4 como el identificador de la secuencia por orden de definición de variables.

3.1.3 Dominio de las Expresiones ($\mathcal{EXP}$)

Los elementos de este dominio son las expresiones que se le asocian a los identificadores de variables y tendrán que ser evaluadas. Pueden ser: expresiones primitivas asociadas a un álgebra y de entrada y salida. Algunos ejemplos:

$E_P \longrightarrow$	$\nu \mid \Delta$	Valor (ver 3.2)
	$\mid E_{P1} + E_{P2} \mid E_{P1} * E_{P2} \mid$	Expresiones aritméticas
	$\mid \mathrm{FUN}(E_{P1}, E_{P2}, \ldots, E_{Pn})$	Funciones predefinidas
$E_E \longrightarrow$	$entrada(\Upsilon)$	Expresiones de entrada
$E_S \longrightarrow$	$E_P; salida(\Upsilon)$	Expresiones de salida

E_P son las expresiones que involucran variables de la Hoja de Cálculo (en sus diversas presentaciones), mientras que E_E y E_S corresponden a expresiones que involucran regiones de la hoja.

3.1.4 Dominio de los Comandos ($\mathcal{CMD}$)

Los comandos se pueden agrupar en comandos de Regiones, Ambientes y de Entrada y Salida. Los de Región permiten definir subconjuntos de variables. Los de Ambientes son aquellos que transforman el ambiente modificando la secuencia de evaluación y permitiendo

la manipulación de los elementos asociados a las variables tal como el editor de expresiones.

Los comandos que se presentan dan algunas facilidades de las Hojas de Cálculo estudiadas, las extienden y generalizan. Como ejemplos, los casos siguientes:

$\Gamma_R \longrightarrow$	DefColec($\Delta_1, \ldots, \Delta_n, \Upsilon$)	Definir región con variables
	| DefPred(Δ, Υ)	Definir región con predecesores funcionales de Δ
$\Gamma_A \longrightarrow$	movAbs(Υ_1, Υ_2)	Mover región en forma absoluta
	| Calcular(o)	Evalúa la sec. en el contexto dado

Hasta este punto, se ha establecido los dominios semánticos y su morfología y se ha dado una descripción informal del significado. En los dos puntos siguientes, se expresará de manera formal el significado de las expresiones y de los comandos.

3.2 Dominios Semánticos

Es aquí donde se definen los valores u objetos matemáticos que serán asociados a los elementos sintácticos del lenguaje (el universo de discurso semántico). Los dominios semánticos que se necesitan para interpretar el lenguaje son los siguientes:

- Valores $\mathcal{VAL}$ ($\nu \in \mathcal{VAL}$)
- Tipos $\mathcal{TIP}$ ($\tau \in \mathcal{TIP}$)
- Ambientes $\mathcal{AMB}$ ($\alpha \in \mathcal{AMB}$)

3.2.1 Valores($\mathcal{VAL}$)

El dominio $\mathcal{VAL}$ está constituido por los valores ó constante de los tipos : lógico (B) , reales ($\Re$), enteros ($\mathcal{Z}$), cadenas finitas del alfabeto (Σ^*) y un valor nulo.
Es decir:

$$\nu \in \mathcal{VAL} = \mathcal{B} \oplus \Re \oplus \mathcal{Z}; \oplus \Sigma^* \oplus nula \tag{6}$$

3.2.2 Tipos ($\mathcal{TIP}$)

El dominio de tipos corresponde a los identificadores de tipos asignables a las variables. Este conjunto es fijo (tipo predefinidos) o puede ser ampliado por los usuarios. En nuestro caso lo trataremos como un conjunto extensible o no. El elemento genérico de este conjunto lo denominaremos τ.

3.2.3 Ambientes($\mathcal{AMB}$)

Es una recopilación de los contextos de variables, región y evaluación definido de la forma siguiente:

$$\mathcal{AMB} = \mathcal{CTXV} \times \mathcal{CTXR} \times \mathcal{CTXE}$$

$$\alpha = < \varphi , \varphi_r , \varphi_o >$$

Donde en el Contexto de Variables ($\mathcal{CTXV}$) se reflejan las posibles asociaciones entre las variables y la tripleta constituida por <valor , tipo , expresión>

$$\mathcal{CTXV} = [\mathcal{VAR} \rightarrow \mathcal{VAL} \times \mathcal{TIP} \times \mathcal{EXP}]$$

$$\varphi \in \mathcal{CTXV} \quad \leftrightarrow \quad \varphi = < \varphi_v, \varphi_t, \varphi_e >$$

En el Contexto de Regiones($\mathcal{CTXR}$) le asociamos a una región, un conjunto de variables y su secuencia de presentación

$$\mathcal{CTXR} = [\mathcal{REG} \rightarrow \mathcal{P}(\mathcal{VAR}) \times \mathcal{VAR}^*]$$

$$\varphi_r \in \mathcal{CTXR} \quad \leftrightarrow \quad \varphi_r = < \varphi_{r_1}, \varphi_{r_2} >$$

Un elemento distinguido del dominio de regiones es Υ_0 que corresponde a la región definida por las variables que se visualizan en la pantalla. La función, φ_r no podrá ser sobreyectiva ya que $\mathcal{P}(\mathcal{VAR})$ tiene la potencia del continuo mientras que los identificadores son numerables.

El Contexto de Evaluación($\mathcal{CTXE}$) es la función que asocia a cada identificador $\mathcal{ORD}$ la secuencia que indica el orden en que serán evaluadas las variables y el modo representando, el modo manual (m) ó modo automático (a)

$$\varphi_o \in \mathcal{CTXE} = [\mathcal{ORD} \rightarrow (\mathcal{VAR}^* \times \{m, a\})]$$

3.3 Funciones de Interpretación

Ahora se definirá en forma precisa lo que sucede cuando se evalua una expresión y se ejecuta un comando. Las funciones de interpretación son:

$$\mathcal{E} : \mathcal{EXP} \longrightarrow (\mathcal{AMB} \longrightarrow (\mathcal{VAL} \times \mathcal{VAR}^*))$$

$$\mathcal{I} : \mathcal{CMD} \longrightarrow (\mathcal{AMB} \longrightarrow \mathcal{AMB})$$

es decir dada una expresión, su interpretación será una función del ambiente a los valores y a los órdenes de evaluación. Dado un comando obtenemos una función de ambientes en ambientes (i.e: la interpretación de comandos es una transformación de ambientes).

La definición de la funciones de interpretación se realiza por casos, utilizando la sintaxis abstracta (3.1). También utilizaremos la notación lambda (λ) para expresar las funciones.

Por razones de espacio solo se dá una muestra de las funciones de interpretación de expresiones y comandos. Una versión ampliada se encuentra en [4].

3.3.1 La función de interpretación ($\mathcal{E}$) de Expresiones

Si se denota como α la colección de los ambientes

$$\alpha = << \varphi_v, \varphi_e, \varphi_t >, \varphi_o, < \varphi_{r_1}, \varphi_{r_2} >>$$

siguiendo las definiciones de las expresiones (3.1.3) resulta

$$\mathcal{E}[\![\nu]\!] = \lambda\alpha \cdot < \nu, \epsilon >$$

$$\mathcal{E}[\![\Delta]\!] = \lambda\alpha \cdot < \varphi_v(\Delta), \epsilon >$$

Esta evaluación corresponde a considerar la expresión asociada a una variable como una función. Los valores asociados a los elementos que aparecen en la expresión al momento de realizar la evaluación son sus parámetros actuales.

3.3.2 Funciones de interpretación ($\mathcal{I}$) de Comandos

A continuación daremos algunos de los comandos del lenguaje. Recordemos que la sintáxis de $\mathcal{I}$ es:

$$\mathcal{I} : \mathcal{CMD} \longrightarrow (\mathcal{AMB} \longrightarrow \mathcal{AMB})$$

y un ambiente α está dado por:

$$\alpha = << \varphi_v, \varphi_e, \varphi_t >, \varphi_o, < \varphi_{r_1}, \varphi_{r_2} >>$$

$$\mathcal{I}[\![\ \text{DefColec}(\Delta_1, \ldots, \Delta_n, \Upsilon)]\!] = \lambda\alpha \cdot < \alpha \downarrow 1, \alpha \downarrow 2, < \varphi_{r_1}[\Upsilon|\Delta_1, \ldots, \Delta_n], \varphi_{r_2}[\Upsilon|\ \text{sec}] >>$$

$$\mathcal{I}[\![\ \text{calcular}(\Theta)]\!] = \lambda\alpha \cdot \text{evaluar} < \alpha \downarrow 1, \varphi_o[\Theta_o|\text{actual}, \alpha \downarrow 3 >$$

Donde: actual es la secuencia de evaluación que obtenemos a partir de $\varphi_o(\Theta)$, haciendo substituciones "convenientemente " [1] a las variables que tenían asociadas expresiones de

[1] Los modos de sustitución serán motivo de estudio ulterior

entrada y salida (ver 3.1.3) y

$$\text{evaluar} = \lambda\alpha \cdot \text{si vacía}(\varphi_o(\Theta_o)) \text{ entonces } \alpha \text{ sino } \mathcal{I}[\![\text{ calcular}]\!] \alpha'$$

donde

$$\alpha' = << \varphi_v[\text{cab}\varphi_o(\Theta_o) \mid \mathcal{E}[\![\varphi_e(\text{cab}\varphi_o(\Theta_o))]\!]\alpha, \varphi_e, \varphi_t >, \varphi_o[\Theta_o \mid \text{cola}\varphi_o(\Theta_o)], \varphi_r >$$

La interpretación evalúa la secuencia de ordenamiento sin interrupciones. Los operadores de cab y cola son los usuales de cadenas. Seleccionar un orden de cálculo (Θ en nuestro caso) no ocurre en las hojas de cálculo. Normalmente es el orden por dependencia funcional (Θ_1), y no explicitado cuando ocurren circularidades.

3.4 Conclusiones

Hemos presentado una descripción formal de una familia de aplicaciones denominadas "Hojas de Cálculo". En esta descripción se establece en forma precisa, concisa y sistemática los conceptos que aparecen en dicha familia de aplicaciones, permitiendonos analizar posibles fallas o semánticas "no ortodoxas" para esos conceptos en algunas aplicaciones.

Pensamos que conceptos tales como: secuencia de evaluación, tratamiento de regiones, manejo de tipos evidencian diferencias en las distintas aplicaciones y hasta en una misma aplicación en distintas plataformas. Por último, disponer de una especificación formal es disponer de una herramienta de verificación y/o análisis.

Referencias

[1] Scott, D.; "Domains for Denotational Semantics", ICALP'82.

[2] Soler, R.;"Fundamentos de un Meta-Lenguaje para Describir Semántica", Dpto. de Computación, U.C.V.,1982.

[3] Meyer, B.;"Introduction to the Theory of Programming Lenguages", Prentice Hall Series in Computer Science, 1990.

[4] Arráiz E.,Soler R.,Zoltan C.;"Análisis de Hojas de Cálculo", a ser publicado, 1992.

GILENA: GENERADOR DE INTERFACES EN LENGUAJE NATURAL

John Atkinson A. Javier Cañas R.

Departamento de Informática
Universidad Técnica Federico Santa María
Valparaíso, Chile

RESUMEN

En este trabajo se describe una herramienta de software llamada GILENA, que permite facilitar la tarea de diseñar interfaces en lenguaje natural.

GILENA se basa en un modelo lingüístico llamado Gramática Sistémica, aunque también es posible usarla en otros modelos debido a que su núcleo principal es un intérprete de ATN.
Se describe el modelo gramatical, sus facilidades, uso y estructura.

La experiencia obtenida con la herramienta se obtuvo de una aplicación práctica que previamente había sido desarrollada con métodos tradicionales. Se pudo constatar que a pesar que se genera una sobrecarga en el software, reduce en un porcentaje muy importante los tiempos involucrados en el ciclo de vida de una aplicación.

1. INTRODUCCION

Las interfaces en lenguaje natural (ILN) resultan atractivas en ambientes tales como: sistemas de consultas, pequeñas bases de datos, programas de aplicación orientados hacia usuarios inexpertos etc...

Lo que no resulta natural, es el diseño de estas interfaces. Diversos factores hacen compleja y tediosa esta tarea: el diseñador se debe relacionar con aspectos computacionales y lingüísticos, y por otro lado, la mantención del software es aún más complicada que el diseño, ya que los usuarios en forma rápida tienden a hacer fuertes exigencias a la interfaz una vez que la dominan.

GILENA es una herramienta que permite automatizar el ciclo de vida de las ILN basadas en enfoques ligüísticos. GILENA es un intérprete de ATN [Wood70] al cual se le han incorporado facilidades de programación. Genera código C y hace posible ligar la interfaz con otras aplicaciones.

2. MODELO GRAMATICAL

La mayor dificultad que presentan las ILN, es que aún en ambientes muy restringidos, el universo lingüístico suele ser bastante extenso. Por esta razón en los últimos años se han realizado esfuerzos para desarrollar formalismos y metodologías que permitan un manejo computacional eficiente del lenguaje.

GILENA tiene su fundamento en la Gramática Sistémica desarrollada por Halliday [Wino73], sin embargo también es posible adaptar GILENA a modelos que usen la Gramática Transformacional y Gramática de Estructura de Frase.

La Gramática Sistémica tiene sus raíces en la antropología y sociología. En este enfoque se considera al lenguaje como una actividad social, razón por la cual el contexto del discurso tiene una gran importancia.
Se considera al lenguaje en términos de distintos estratos: fonológicos, morfológicos, sintáctico y semántico. El análisis completo del discurso requiere del análisis en los diferentes estratos, pero como existe una fuerte relación entre ellos, en la comprensión de oraciones por parte del computador, basta considerar el nivel sintáctico y semántico.

En el nivel sintáctico se hace una nueva división en tres niveles: la palabra, los sintagmas y las oraciones. Las palabras se agrupan en clases de acuerdo a su categoría léxica (nombres, verbos, preposiciones, etc...), los sintagmas se agrupan en categorías sintácticas (nominal, verbal, preposicional).

Para ilustrar este método de análisis se considera en la Fig Nº1 la oración: "El futbolista murió en el hospital"

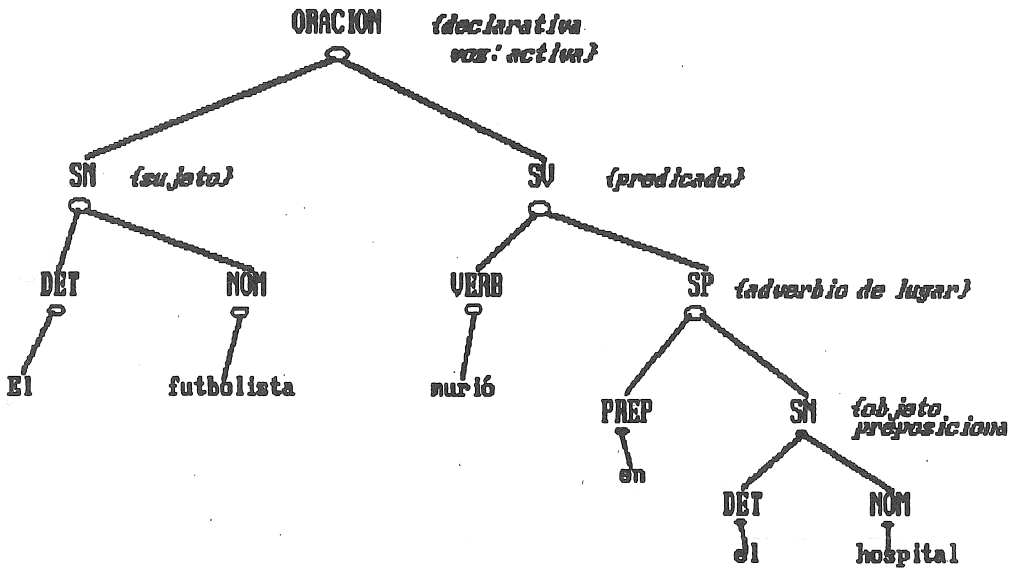


FIG Nº1

La estructura arbórea muestra los diferentes niveles del análisis, es decir palabras, sintagmas y oraciones. GILENA permite a partir de una oración o frase de entrada, generar este tipo de estructura para, a partir de su conocimiento sintáctico a través de este análisis, obtener información semántica.

3. PRESENTACION DE GILENA

En vez de definir formalmente el lenguaje GILENA, lo presentaremos a través de ejemplos.

Como primer ejemplo, consideremos el lenguaje:

$$L = \{a^n b^n c^n / n >= \emptyset\}$$

El ATN que reconoce palabras válidas de L se muestra en la figura Nº2.

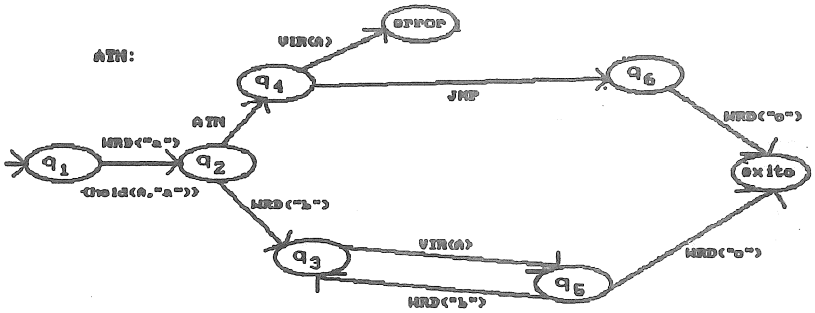


FIG N°2

El estado inicial es q1. El arco rotulado WRD("a") verifica si el primer simbolo leido corresponde al simbolo "a". En caso positivo consume el simbolo, pasa al estado q2 y como efecto lateral guarda el simbolo "a" en el stack A. Esto se realiza mediante la acción hold.
Si el próximo simbolo es "b", se pasa a q3 consumiendo el simbolo. En caso contrario se activa el arco rotulado ATN cuyo efecto es invocar recursivamente al mismo ATN.
El arco VIR(A) hace un POP del stack global A y en caso que no esté vacio, pasa al próximo estado. El arco JMP pasa incondicionalmente al próximo estado.
Una conclusión importante que muestra este ejemplo es que el poder computacional de un ATN es equivalente a una máquina de Turing ya que el lenguaje L es sensible al contexto.

La expresión de GILENA de este ATN es :

```
ATN:
        q1 -> WRD("a") {hold(A,"a");} to q2;
        q2 -> ATN to q4 |
              WRD("b") to q3;
        q3 -> VIR(A) to q5;
        q4 -> VIR(A) to error |
               JMP to q6;
        q5 -> WRD("c") to exito |
               WRD("b") to q3;
        exito -> ;
        q6 -> WRD("c") to exito;
        error -> ;
enda
```

Se aprecia la estrecha relación entre el ATN y el correspondiente código en GILENA.

Estructura del Lenguaje

El lenguaje consta de las siguientes 25 palabras reservadas:

dictionary	special	enda
end	symbols	to
structures	useless	jmp
ends	inseparable	cat
type	conjuction	wrd
endt	separator	vir
categories	finish	integer
endc	atn	word
single		

El lenguaje se divide en 5 áreas principales:

```
I     < Declaraciones del usuario: definiciones
        en lenguaje C >
      dictionary
II    < Definiciones del diccionario >
      endd

      special symbols
III   < Especificación de símbolos especiales >
      ends

      atn
IV    < Definición del ATN >
      enda

V     < Procedimientos definidos por el usuario >
```

El área I es opcional. Las variables definidas por el usuario deben estar escritas en código C.
El área II especifica el diccionario de palabras. Se encuentra dividido en categorías sintácticas y además a cada palabra se le asocia sus rasgos de acuerdo a la categoría.
Esta área también es opcional.
Como ejemplo se da un diccionario que incluye las categorías sintácticas verbo, nombre sustantivo y clítico:

```
dictionary
      structures
            type est_verbo
                  tiempo = (pas, pte, fut)
                  numero = (sing, plur, neu)
                  persona = integer
                  modo = word
            endt
            type est_sust
                  tipo = (comun, propio)
                  genero = (mas, fem, neu)
                  numero = (sing, plu, neu)
                  persona = integer
                  humano = (si, no)
            endt
      ends
      categories
            verbo "verbos.cat" type est_verbo
            sust "sustan.cat" type est_sust
            clitico "clitic.cat" type single
      endc
endd
```

Las palabras mismas están almacenadas en los archivos: verbos.cat, sustan.cat y clitic.cat.
Una entrada del archivo verbo.cat podría ser:

listar, pte, sing, 1, indicativo

Una entrada del archivo sustan.cat podría ser:

juan, propio, mas, sing, 3, si

El archivo de clíticos no tiene estructura definida. Por ejemplo:

me
se
le

Es importante considerar que al revisar el diccionario, GILENA verifica si la categoría almacenada es compatible con la estructura definida en él.

El área III es el área de símbolos especiales. Estos son los símbolos que al no poder ser manajados por el diccionario, merecen un trato especial. Un ejemplo sería:

```
special symbols
      useless (por favor, tenga la amabilidad)
      inseparable (tarjeta de crédito; TC; sustantivo)
      conjunction (y, e)
      separator " ;:.!"
      finish (me voy)
ends
```

El Area de símbolos especiales es obligatoria. Esto es así porque la sección "finish" se debe incluir indicando la secuencia que provocará el término de la sesión. Las demás secciones son opcionales.
Los símbolos "useless" son secuencias de palabras que no aportan información semántica. Al marcarlos como "useless" son eliminados del proceso de análisis.
Los símbolos "inseparable" son secuencias de palabras que por su naturaleza deben ser consideradas como un todo. Por ejemplo: Juan Pérez, Tarjeta de Crédito etc...
GILENA contiene funciones predefinidas que permiten un tratamiento automático de conjunciones. Estas funciones son activadas por aquellos símbolos que están declarados como conjunciones.
Los símbolos "separator" son aquellos caracteres que se usarán como separadores de palabras.

El Area IV es el núcleo principal de GILENA porque especifica el ATN correspondiente a la gramática del lenguaje. Un segundo ejemplo de ATN correspondiente a una gramática arbitraria se muestra en la figura Nº3:

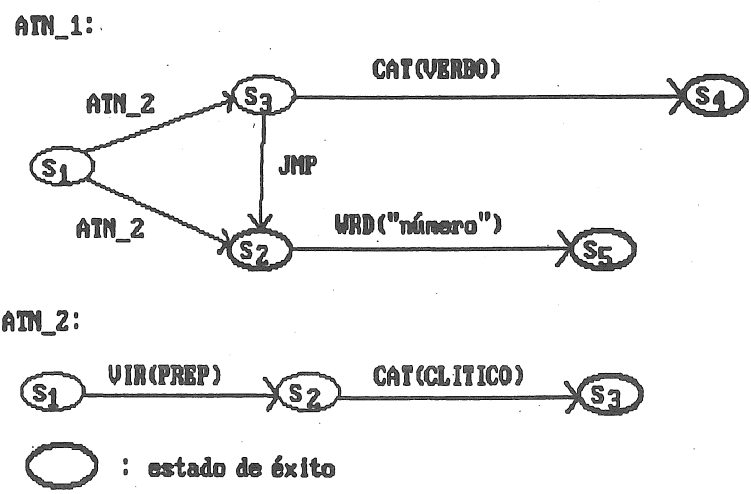


FIG Nº3

La especificación en GILENA de esta red es:

```
atn
      atn_1 :
                  s1 -> atn_2 to s3 |
                        atn_2 to s2
                        ;
                  s2 -> wrd("número") to s5 |
                        ;
                  s3 -> cat(verbo)
                        {
                        /* ejemplo de efectos
                           laterales */
                        printf("atnword vale ");
                        printf("%s\n",atnword);
                        analice(atnword);
                        } to s4 |
                        jmp to s2
                        ;
                  s4 -> ;
                  s5 -> ;
      &
      atn_2 :
                  s1 -> vir(prep) to s2 ;
                  s2 -> cat(clitico) to s3;
                  s3 -> ;
enda
```

El área V es la llamada área de procedimientos del usuario. Su finalidad es permitir a los usuarios desarrolladores, escribir sus propios procedimientos en lenguaje C.
Existe un procedimiento predefinido llamado atnparse, el cual es encargado de administrar y controlar el avance del ATN. La definición de atnparse es:

```
int atnparse(frase_entrada)
char *frase_entrada;

retorna:
      -1: si hubo error sintáctico
      +1: si se analizó correctamente la frase
       2: si se reconoció el símbolo final de la
          sesión
```

Una llamada típica tiene la siguiente estructura:

```
atn
........
enda

main()
{
char frase[80];
int status;
      do {
            printf("Ingrese Consulta: ");
            gets(frase);
            status = atnparse(frase);
            switch (status) {
                  case -1 : Error();
                  case 1  : generar_respuesta();
                  }
      } while (status != 2);
      printf("Fin sesión ..\n");
```

4. USO DE GILENA Y FACILIDADES DE DEPURACION

GILENA provee dos modos de entrada: a través de un archivo o de la entrada estándar. Si se especifica un archivo el formato es:

gilena [-t] <arch_especif>

la opción -t, indica que se debe generar una traza para la ejecución de la interfaz, y <arch_especif> corresponde al nombre del archivo de especificación de entrada.
Bajo cualquier condición aparece en la salida estándar mensajes que indican las áreas de análisis:

```
Analizando área diccionario.......
Analizando área de símbolos especiales.......
Analizando área de ATN.........
Compilación OK......
```

GILENA es una herramienta para ser utilizada por desarrolladores de interfaces, por esta razón se le ha incorporado facilidades para realizar trazas del recorrido del ATN en el análisis de las oraciones.
La traza se indica con la opción -t. La traza se utiliza para el recorrido de los arcos del ATN. Para avanzar al siguiente estado en esta modalidad, el usuario debe responder 'y' para continuar o 'n' para finalizar. Como resultado del recorrido de un arco se puede obtener: TRUE si la condición es verdadera, FAIL si falla y REDO si la condición falla y se debe realizar un backtracking.

Existen una serie de mensajes antes de atravesar cada arco. A modo de ejemplo, parte de una sesión con el ATN de la figura Nº3 sería:

```
Starting ATN atn_1
PUSHing ATN atn_1 to state s1; Continue (y/n)? y
      in state s1
PUSHing ATN atn_2 to state s1; Continue (y/n)? y
      in state s1


.................
Trying an WRD arc to state s5, expecting ("número");
Continue (y/n)? y
..............
```

5. EXPERIENCIA CON GILENA

Para probar GILENA como herramienta de desarrollo de interfaces, se realizó una aplicación que consistió en un sistema de consultas automatizadas para clientes de un Banco. Se eligió esta aplicación porque ya existía experiencia previa en este universo lingüístico a través de desarrollos en C, Lisp y PROLOG.

No se hicieron mediciones cuantitativas de ingeniería de software, pero el tiempo de desarrollo fue sustancialmente menor. Si bien el código generado es más grande que la versión escrita en C, una vez diseñada la interfaz resultó muy fácil introducir modificaciones y ampliar el universo.

Una sesión típica con esta interfaz es:

C) Por favor, me puede dar mi saldo total
R) Su saldo total es $856.987

C) Cuáles son los últimos cheques cobrados?
R) Los 4 últimos cheques cobrados son: 7Ø91, 7Ø95,
   7Ø92, 7Ø9Ø

C) Saldo?
R) Saldo total o disponible?
C) Disponible y total!
R) Su saldo disponible es $75Ø.432 y el total es $856.978

etc...

donde C) indica la consulta del cliente y R) la respuesta de la interfaz.
De este ejemplo se puede visualizar también que si bien las consultas son simples, involucran aspectos lingüísticos complejos como tratamiento de elipsis y conjunciones.

6. DISEÑO DE GILENA

La arquitectura de GILENA se puede ver en la figura Nº4:

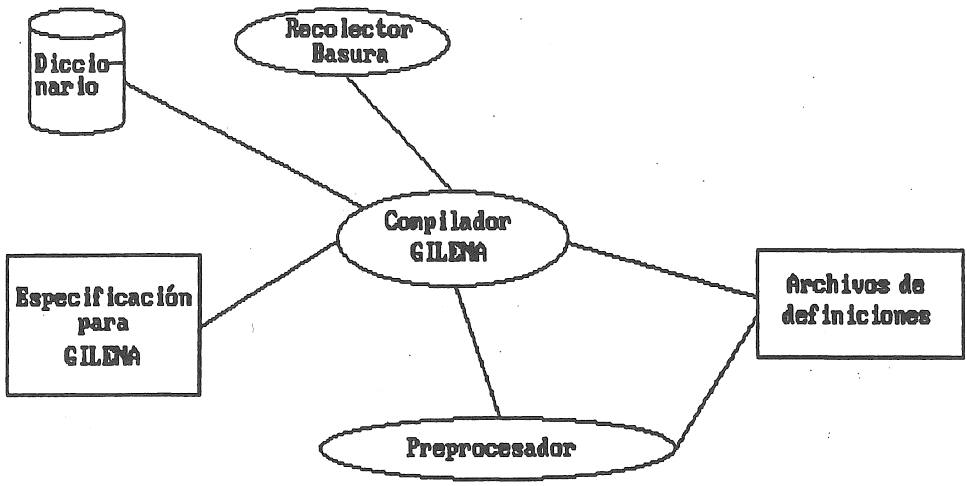


FIGURA Nº4

El núcleo de esta arquitectura es el compilador GILENA. Este compilador es el encargado de recibir el archivo de especificación, analizarlo y generar el código fuente para la interfaz.
El compilador GILENA genera código C. Por esta razón necesita información de librerías y de archivos de definición.
El preprocesador tiene por función reemplazar secuencias de símbolos para facilitar la tarea del compilador.
El recolector de basura corresponde a un módulo que contiene las funciones que permiten optimizar el uso de la memoria.
El diccionario son archivos ASCII, uno por cada categoría sintáctica para facilitar su mantención.

7. CONCLUSIONES

El diseño y construcción de GILENA forma parte de una línea de investigación del Departamento de Informática de la Universidad Técnica Federico Santa María.
GILENA es el resultado de la experiencia lograda a través de varios proyectos en los cuales se experimetó con varias metodologías, ambientes y lenguajes.
La conclusión más relevante es que como herramienta es útil y

permite una fácil experimentación con gramáticas. Como el desarrollo se realizó en lenguaje C se hicieron también experiencias de portabilidad entre estaciones SUN y computadores personales. La experiencia en el desarrollo y utilización de GILENA también permitió detectar sus debilidades. Aspectos de definición del lenguaje y eficiencia del compilador se han considerado para una segunda versión.

REFERENCIAS BIBLIOGRAFICAS

[Aho86] Aho Alfred, Ullman Jefrey, "Compilers: Principles, Techniques and Tools", Addison-Wesley, 1986

[Bolc86] Bolc Leonard, "The Design of Interpreters, Compilers and Editors for Augmented Transition Networks", Springer-Verlag, 1983

[Caña85] Cañas Javier, "Interface en lenguaje natural para consultas interactivas", Universidad de Chile, 1985

[Cis78] Cisternas Ricardo, " Creación y evaluación de métodologia para construcción de interfaces en lenguaje natural", Universidad de Chile, 1987

[Yang9Ø] Yang Gi-Chul, "On deterministic control of Augmented Transition Networks", X conferencia Chilena de Ciencias de la Computación, Chile, 199Ø

[Wino73] Winograd Terry, "Research in Natural Languages", Artificial Intelligence Laboratory, MIT, Mayo 1973

[Wood7Ø] Woods W, "Transition Networks Grammars for natural language analysis", Communications of the ACM, Octubre 197Ø

UN MODELO DE EVALUACION DEL DESEMPEÑO DE SISTEMAS CLIENTE-SERVIDOR*

Francisco Aurtenechea; Gabriel Rosenberg; Ignacio Casas

Pontificia Universidad Católica de Chile
Depto. de Ciencia de la Computación
Casilla 306, Santiago, Chile
FAX: (5-62) 5524054 email: fa@puc.cl

RESUMEN

En este trabajo se presentan metodologías y herramientas para la evaluación y predicción del desempeño de aplicaciones cliente/servidor a través de redes locales Ethernet. El objetivo es evaluar el impacto de los distintos niveles de desempeño de la red en el desempeño global de este ambiente transaccional. Se presenta el desarrollo de herramientas de comunicación como medio ambiente de monitoreo y medición que, junto a modelos analíticos de desempeño, permitan proyectar el impacto de los diferentes niveles de servicio de la red en aplicaciones cliente-servidor. Parte de esta metodología es evaluada mediante mediciones y evaluación del desempeño en el acceso de clientes a una aplicación de base de datos remota.

Palabras Claves: Sistemas Cliente-Servidor, Modelación Analítica, Redes Locales.

I. INTRODUCCION

El auge de las redes locales, se ha traducido en un amplio desarrollo de herramientas de comunicación tendientes a proveer un medio ambiente de operación a estaciones de trabajo. Surgen los sistemas operativos distribuídos [6,15] y esquemas de comunicación cliente/servidor [2,3,15], los que permiten hacer un uso más efectivo de los recursos de la red.

La mayor cobertura de acceso a los recursos distribuídos, causa, además, que la demanda por los recursos de la red, tanto de hardware como de software tengan un aumento sostenido. El monitoreo, evaluación y predicción del desempeño es de vital interés en la evolución de este ambiente computacional. En la medida que los sistemas distribuídos evolucionen y la demanda de uso aumente, el estudio de su desempeño es importante como elemento de diseño y planificación de capacidad.

Hasta la fecha se han desarrollado diversos modelos tanto analíticos como de simulación, para predecir el comportamiento de una red, en función del tráfico o demanda de acceso en la misma [4,5,7,8]. Dichos modelos, en general, analizan aspectos de bajo nivel en la red, sin presentar esquemas comparativos del impacto del tráfico en las distintas interfaces de la red.

En este trabajo se plantea el problema de desagregar la evaluación del impacto de los distintos niveles de interacción de la red en el desempeño de sistemas cliente/servidor. Se proponen metodologías y herramientas de comunicación para el monitoreo y predicción del desempeño de sistemas de acceso distribuído en red local.

Se describe la construcción de un modelo de comunicación, basado en un servidor de aplicaciones [2], tanto para canalizar el acceso, a clientes, a aplicaciones de la red, como para medir el desempeño en las interfaces de comunicación. Mediante la activación controlada de tráfico, se dispone de un monitor de red activo para conocer los retardos en distintos niveles de la red en función del patrón de demanda de aplicaciones específicas.

* Este trabajo fue financiado por FONDECYT, proyecto 0771/91

Se plantea la construcción de modelos de redes de espera [9] para predecir el desempeño de las aplicaciones en distintos escenarios tanto de configuración como de tráfico. Finalmente, se presenta un análisis preliminar de desempeño, mediante la realización de diversas mediciones de tiempos en el acceso a una aplicación remota, en función del número de clientes.

II. ELEMENTOS DE DESEMPEÑO

Cuando dos nodos se comunican mediante una red, se produce el flujo de transacciones, siendo cada una un mensaje de consulta (desde un cliente) seguido de un mensaje de respuesta (desde un servidor). El valor medio del tiempo de respuesta depende de diversos factores tales como configuración, tráfico y estrategias de diseño. Entre estos factores se incluyen: los medios de comunicación, la topología de la red, los protocolos de comunicación, la capacidad de buffers, velocidades de los procesadores, etc.

El desempeño global de las aplicaciones en una red depende de estos factores y de su mutua interacción. La identificación adecuada de estos factores constituye un requisito indispensable si se desea establecer el desempeño integral de las aplicaciones de la red, junto con la obtención de los factores críticos que afectan su comportamiento

En una red local, los protocolos de comunicación corresponden a una estructura de múltiples niveles que permiten el intercambio de datos entre sus nodos. Cada nivel soporta funciones específicas y a través de interfaces se enlaza con niveles superiores hasta alcanzar las aplicaciones de nivel de usuario. Con respecto a este nivel de comunicación, es necesario conocer:

- El efecto en el desempeño de la red debido a las estrategias de almacenamiento y protocolos de transferencia a través de los diversos niveles funcionales que soportan el diálogo entre clientes y servidores.
- La significación de que cierto procesamiento y actividades de comunicación puedan ser realizadas por hardware o procesadores dedicados (p.ej. en el control de protocolos de nivel intermedio como TCP/IP).
- La importancia de dedicar mayor número de procesos de comunicación a las actividades de transferencia de datos, tanto a nivel de cliente como servidor.

Por otro lado, el uso de un gran ancho de banda en la transmisión de bits seriales es uno de los atractivos de las redes locales. A las técnicas en uso actualmente (p.ej. par trenzado, cable coaxial), se están incorporando otras con un mayor ancho de banda disponible, tal como CATV y fibra óptica. Por lo tanto es necesario establecer lo siguiente:

- Cuan significativo es el ancho de banda en el retardo total de una transacción en una comunicación interprocesos a través de la red.
- Cuan sensible es el tiempo de acceso a la red al tamaño de la misma, al tamaño de los paquetes, al tráfico. Cuan importante es este tiempo en el desempeño global de las aplicaciones de red.
- El efecto de ciertas estrategias de control de flujo en la red (p.ej. el uso de *acknowledgements*) en el retardo total de cada transacción.

Finalmente, es necesario considerar factores que son independientes de la red, tal como el tipo de aplicaciones usadas a través de la red, la carga en los servidores, la configuración de los "hosts" (cantidad de memoria, velocidad de dispositivos, etc). La eficiencia de los protocolos de bajo nivel no garantiza un tiempo de respuesta adecuado. Una solicitud a un servidor compartido puede esperar a que otros procesos sean atendidos, o la disponibilidad de controladores de disco, etc. Este retardo puede ser significativo si la carga en la red aumenta. Entonces, es relevante establecer:

- La diferencia de velocidad entre los datos entregados al "host" y la velocidad a la cual éste puede absorberlos.

- Hasta donde puede ser mejorado el desempeño de las aplicaciones, al modificar ciertas estrategias de transferencia de datos (p.ej. uso de buffers dedicados) tanto a nivel de servidor como de cliente.
- Para una aplicación típica, cuan significativo es el tiempo de procesamiento de los datos en la red y en los protocolos, comparado con las actividades de lectura/escritura de datos en el servidor.

Estos y otros factores se deben considerar para proveer una perspectiva más global al evaluar el desempeño de sistemas de acceso distribuído. Es difícil, sin embargo, discernir acerca de todos los factores que influyen en este desempeño, debido a la variabilidad de configuraciones y patrones de tráfico que pueden existir. En la medida que sea posible aislar la mayor cantidad de estos factores, mayor será la posibilidad de conocer y predecir el desempeño de estos sistemas en su dinámica de evolución permanente.

III. INFRAESTRUCTURA DE EVALUACION

Un usuario conectado a un computador de la red es un cliente remoto para ese computador, el cual destina un proceso (servidor) para atender los requerimientos del cliente. Con la incorporación de sistemas operativos de red, sistemas cliente/servidor, etc, se pone a disposición de los usuarios los recursos de la red en forma casi transparente. La pérdida de transparencia estaría asociada a la contención de la red.

Para medir el desempeño de las transacciones de red en este ambiente de interacción, se desarrolló un sistema servidor de aplicaciones [2]. Esta es una herramienta transparente de acceso a los recursos de la red, e incorpora mecanismos de medición del desempeño en distintos niveles de interacción (interfaces de comunicación) entre un cliente y una aplicación remota.

Estos módulos de comunicación están disponibles tanto en equipos PC-DOS como en equipos Unix de la red (Ultrix y System V). Se utiliza como plataforma de comunicación, una red Ethernet [11], el protocolo TCP/IP y "socket library", permitiendo conectar procesos remotos entre un cliente, asociado a un usuario remoto, y un servidor, asociado a una aplicación remota.

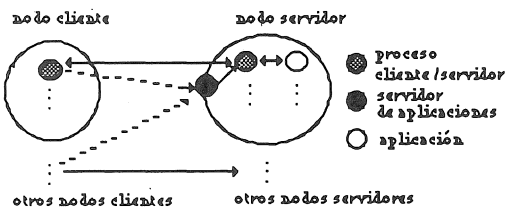


Figura 1. Servidor de aplicaciones

Cuando el servidor de aplicaciones recibe una solicitud de conexión, crea un subproceso y queda en espera de otro cliente, permitiendo a varios clientes interactuar con una aplicación de cualquier nodo de la red. Cada subproceso recibe el nombre del servicio requerido y lo ejecuta, comunicando el "input-output" de dicho servicio con el proceso cliente en el nodo remoto (ver fig.1).

Cada subproceso servidor crea un canal de comunicación para dar acceso, a clientes, a aplicaciones remotas, y re-direcciona la salida de ésta en un "buffer" para ser enviada al cliente remoto (ver fig. 2).Así, cualquier aplicación remota se ejecuta en forma transparente, y manteniendo el *overhead* al mínimo posible en el subproceso servidor.

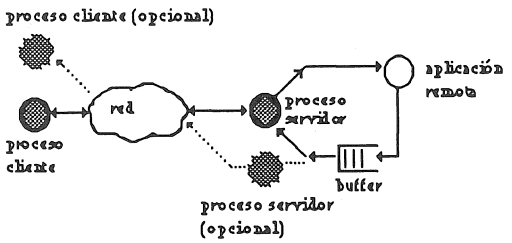


Figura 2.Conexión Cliente/Servidor

IV. METODOLOGIA DE EVALUACION

La realización de evaluaciones sobre un cierto número de sistemas que interactúan, requiere conocer cada nivel de desempeño. El grado de especificación de cada nivel debe estar en relación a la significación que tiene en el desempeño de los sistemas en niveles sucesivos de interacción.

Si el sub-sistema a evaluar es complejo, entonces se puede reproducir sus niveles de desempeño mediante especificaciones funcionales (i.e. modelos analíticos) de las operaciones más relevantes. El uso de modelos simples para representar sistemas complejos es bastante utilizado en evaluación del desempeño de sistemas computacionales [1]. La metodología que se propone para reproducir los niveles de desempeño en la interacción cliente/servidor, se basa en mediciones directas, en el control del tráfico en la red, y en la generación de modelos analíticos.

Las mediciones involucran la inserción de controladores de eventos en el sistema servidor de aplicaciones, tanto en nodos clientes como en nodos servidores. El control de tráfico presupone la posibilidad de generar patrones de carga tanto en la la red como en el servidor de aplicaciones, y que no existan distorsiones por carga externa (i.e. no controlada). Los modelos analíticos se aplican en sub-sistemas que son difíciles de medir directamente o bien que son simples de deducir por esa vía (p.ej. CSMA/CD). Gran parte de la metodología de evaluación requiere de la realización de mediciones en cada nivel de interacción relevante. Las mediciones deben entregar información que permitan identificar los diferentes componentes del tiempo de respuesta de la aplicación y el *overhead* que se produce por utilizar un esquema descentralizado a través de una red de comunicación.

Esta identificación se logra incorporando "relojes" tanto a nivel de cliente como de servidor que permitan conocer los retardos entre cada interfaz de comunicación, por ejemplo: a través del software de comunicación tanto del cliente como del servidor, a través del protocolo de comunicación de la red (p.ej. TCP/IP), en la transmisión de datos por la red, en el servicio local otorgado por el servidor, etc.

Dada la necesidad de acoplar mediciones desde diferentes nodos (cliente y servidor), en tiempo real, se debe sincronizar los relojes de cada uno, o en otras palabras conocer el desfase entre cada uno. Para ello, para iniciar una medición, se propone el siguiente algoritmo: (i) Se almacena la hora que registra el cliente (t_1) y se envía una señal al servidor, (ii) El servidor recibe la señal, y retorna de inmediato al cliente la hora de recepción (t_2) de dicha señal, medida en el servidor, (iii) El cliente recibe la respuesta y registra la hora de llegada (t_3). El desfase de los relojes se puede calcular como: $D = t_2 - (t_3 + t_1)/2$. El retardo para solicitar al sistema el tiempo transcurrido, a través de funciones del sistema operativo, es en general despreciable. Si no fuera así, este *overhead* debe ser incorporado al algoritmo de sincronización.

Para evitar resultados poco representativos, este procedimiento de sincronismo debe ser iterado un cierto número de veces, de manera de tener un desfase de tiempo promedio donde se minimicen las situaciones particulares al efectuar la medición. Así, es posible obtener, tanto para el cliente como para el servidor, un registro de los eventos medidos y los tiempos en los que ocurrieron. Finalmente estos registros pueden ser utilizados por un procedimiento que permita secuencializar los resultados para su interpretación.

Es posible identificar dos grupos de aplicaciones. El primero consiste de aplicaciones que no requieren interacción constante con el usuario (i.e. hay en un pedido inicial y su consiguiente respuesta desde el servidor). Estas aplicaciones son típicas en invocación de comandos remotos. El segundo grupo de aplicaciones, está formado por aquellas en las cuales existe una iteración del tipo solicitud/respuesta, entre el usuario y la aplicación (servidores de archivos, compiladores, sistemas de bases de datos etc.).

El desempeño de una red y de los sistemas componentes depende, además de los aspectos de configuración, del tráfico o carga involucrada. Utilizando el sistema servidor de aplicaciones mencionado anteriormente, se ha implantado un sistema generador de tráfico para incorporar distintos patrones de carga sobre la red en general, y sobre un nodo servidor en particular.

En cada nodo de la red hay un proceso denominado servidor de tráfico, que puede ser invocado desde cualquier cliente, recibiendo el patrón de tráfico a ser generado por el nodo donde se localiza este servidor. Así, el servidor de tráfico puede generar un número arbitrario de procesos clientes, cada uno conectado a un sub-proceso en el servidor de aplicaciones del nodo correspondiente. Mediante un monitor pasivo de red [13] es posible recolectar información del tráfico y retardos sobre la red, y de este modo incorporar los retardos del controlador de la red en los modelos de redes de espera que se propone generar (o bien validar los modelos analíticos de comportamiento de la red, en este caso Ethernet).

Con las mediciones se propone alimentar modelos del sistema, basados en modelos de redes de espera [9], para predecir resultados bajo diferentes valores de variables claves tales como: tipo de carga, velocidad de las CPUs, etc. Además, el sistema servidor de aplicaciones puede ser manipulado modificando elementos de configuración (cantidad de buffers, número de procesos dedicados, etc), para disponer de una mayor variedad de opciones en los modelos de proyección.

V MODELACION DEL SISTEMA

En la construcción y uso de un modelo de red de espera [9], se incluyen, generalmente, tres etapas, a saber: i) Crear el modelo Base, ii) validar el modelo Base, y iii) utilizar el modelo para proyecciones futuras.

Para crear el modelo es necesario medir el sistema computacional, típicamente utilizando un monitor del sistema. Se debe efectuar una medición por un lapso de tiempo suficiente como para que incluya ya sea períodos de alta carga del sistema o alguna condición o proceso definido que se desea monitorear en forma exhaustiva. Debe incluirse en este monitoreo valores de las variables más relevantes del sistema, como uso de CPU y discos, número de ejecuciones de cada proceso (o clase de proceso), terminales activos, etc.

Deben ser definidos los componentes del modelo, identificando los centros de servicio que se desea contenga el modelo. Estos corresponden normalmente a la (o las) CPU, los discos y de ser necesario un centro que represente la acción del controlador de red para representar la red de comunicación, entre otras cosas.

Deben se definidas las clases de procesos. Una clase de procesos consiste de "jobs" que tienen un origen común y que poseen consumos de servicio similares. Para cada clase de proceso se define el consumo promedio de una instancia del proceso en la CPU y en cada disco, en base a las mediciones del monitor. Finalmente, los datos entregados por el monitor se utilizan para dar valores a los parámetros de los componentes del modelo.

Para validar el modelo, los parámetros anteriormente identificados se pueden utilizar para alimentar un software de evaluación de redes de espera, por ejemplo MAP [12]. Las estimaciones de este software son comparadas con las mediciones del monitor y el modelo es validado o rechazado, caso en el que se debe volver a evaluar las clases y los valores de los parámetros. Luego, el modelo se puede usar para realizar proyecciones variando las características de algunos de sus componentes (p.ej. la velocidad de la CPU).

Al proyectar el comportamiento de cierta aplicación, un parámetro de utilidad es el tiempo de respuesta promedio al invocar la aplicación, bajo las condiciones "ambientales" de equipamiento y cantidad de usuarios (clientes) que están invocando la aplicación de forma concurrente. Otra variable importante, como parámetro de

entrada al modelo, es la demanda de CPU de la aplicación. Se considera una demanda por CPU "repartida", ya que hay requerimientos de CPU en los diferentes componentes del software cliente/servidor. Esta demanda por CPU puede ser utilizada para predecir la respuesta de la aplicación en otros ambientes, por ejemplo, si la velocidad de CPU del servidor se duplica.

Tanto el proceso cliente como el proceso servidor demandan un mínimo acceso a disco. Sin embargo, si la aplicación invocada es intensiva en lectura/escritura desde disco, las características de este dispositivo, como tiempo de "latencia" o búsqueda ("seek"), deben ser incorporadas como parámetros del modelo. En caso de existir mecanismos para almacenamiento temporal de información, como "cache disk" (caso típico de servidores de archivos) puede ser necesario incorporar esta información en el modelo.

5.1 Componentes del Modelo

Los componentes que debe incluir este modelo son básicamente los siguientes (ver fig. 3): (1) Procesos (o subprocesos) Clientes, (2) Procesos (o subprocesos) Servidores (3) Protocolos de Red. (p.ej. TCP/IP) (4) Red de comunicación (p.ej. Ethernet), (5) Aplicación invocada (p.ej.. sql, nfs).

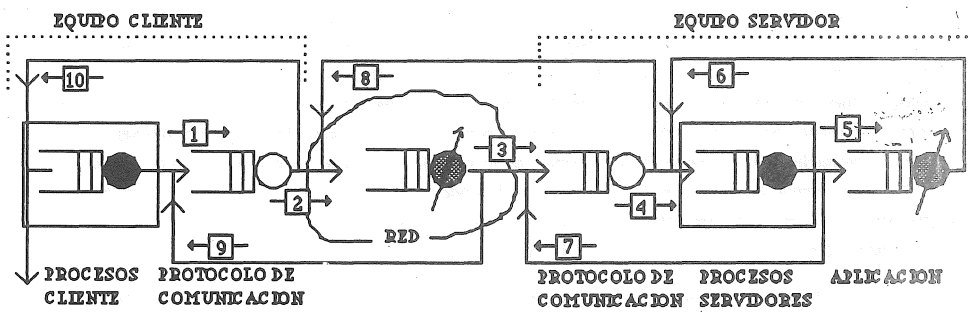


Figura 3. Modelo de red de espera y flujo de datos

Los "pasos" a representar en el modelo corresponden al flujo de datos entre los distintos componentes del sistema cliente/servidor. Dado que se requiere determinar el impacto de estos componentes en el tiempo de respuesta total, el flujo de datos incluye una transacción completa entre cliente y servidor (ver. fig. 3).

El paso 1 indica que un requerimiento (paquete) es enviado desde un cliente hacia la red, pasa por los protocolos de comunicación, en donde se le agrega un encabezado con las direcciones fuente, destino y otra información relevante para el protocolo. El paso 2 corresponde a la transmisión del paquete por la red, y el paso 3 al ingreso del paquete al protocolo de comunicación en el servidor. En el paso 4 el requerimiento es recibido por el proceso servidor y en el paso 5 la aplicación lo recibe y procesa. El mensaje de respuesta efectúa la secuencia inversa: enviado desde la aplicación hacia el subproceso servidor (paso 6), enviado al protocolo de comunicación del servidor (paso 7), transferido por la red (paso 8), procesado en el protocolo de comunicación del cliente (paso 9), y finalmente recibido por el proceso cliente y enviado al usuario (paso 10).

5.2 Procesos Cliente y Servidor

Para el proceso cliente se debe considerar tanto el tiempo de ejecución como la demanda de CPU que este proceso consume. El modelo debe reflejar las variaciones en los tiempos de respuesta, en función del número de clientes, n, el que se puede representar como una tasa de llegada. Para una aplicación, cada cliente pertenece a la misma clase de proceso; luego, si t es la demanda de servicio de un cliente, entonces $\lambda=n/t$ es la tasa de clientes/seg.

La invocación de la mayoría de las aplicaciones es efectuada mediante un requerimiento simple, a través de un paquete de datos. En este caso, la tasa de llegada es igual a la tasa de requerimientos por la red [paquetes/seg]. El tamaño de los paquetes enviados por la red es conocido por lo que se puede calcular cuanto aportan las aplicaciones a la carga de la red (paquetes/seg.* tamaño paquete / capacidad de transmisión). Teniendo acceso, además, a un monitor de redes [13] se puede conocer la carga total de la red, y obtener por lo tanto, el retardo a través de ella (ver 5.3).

El proceso servidor es similar al proceso cliente, existiendo un proceso servidor por cada proceso cliente remoto. Considerando el principio de conservación del flujo, la tasa de llegada de requerimientos al cliente es similar a la del servidor.

5.3 Modelo Representativo de la Red Ethernet

El desempeño de redes Ethernet ha sido extensivamente analizado [1,11,14]. Basado en la teoría de colas y en procesos "markovianos", se puede determinar el tiempo de servicio de un paquete en la red (retardo desde que un paquete es generado por el adaptador de la red hasta que llega al adaptador destino) en función del tráfico en la red. La carga es expresada como una proporción de la capacidad de transmisión de la red (ρ).

El tiempo se considera dividido en intervalos ("slots") cuya duración, S, es equivalente al tiempo de propagación ida-vuelta del canal. (tiempo requerido para que una colisión sea detectada por todas las estaciones). Si hay n estaciones activas (carga instantánea) y ninguna estación transmite, el "slot" es perdido; si sólo una estación transmite, adquiere el canal hasta que finalice de enviar el paquete. Si dos o más estaciones transmiten simultáneamente, ocurre una colisión y el intervalo es perdido. Si C es la capacidad de la red [bits/seg], y P el tamaño promedio de los paquetes [bits], el tiempo de servicio de un paquete, es igual al tiempo de transmisión (P/C) más el tiempo de contención, W. El primero no depende del tráfico, pero el último depende del número de paquetes en espera (carga instantánea), y es igual a la duración del *slot* multiplicado por el número de *slots* dedicados a contención ($W=S*(1-(1-1/n)^{n-1})/((1-1/n)^{n-1})$) [11].

Analizando la red como un modelo de Markov, la carga instantánea aumenta con la tasa de llegada de paquetes (cuyo valor medio es $\rho C/P$) y disminuye con la tasa de envío de los paquetes a destino. Esta es igual a capacidad de transmisión de la red, C/P, multiplicado por su eficiencia, E, cuando la carga instantánea es n, y es igual a $1/(1+ C/P*W)$. El tiempo medio de servicio de un paquete se obtiene (ecuación de Little), tomando el cuociente entre la carga instantánea promedio (número de paquetes en espera de servicio) y la tasa promedio de llegada de paquetes, ρ C/P.

Para modelar esta red es posible utilizar este desarrollo para representarla como un Servidor de Flujo Equivalente (FESC) [9], dependiente de la carga (ρ) de la red y las características de la misma (p.ej. largo de la red, tamaño de los paquetes). Esta FESC puede ser validada mediante monitores de red, midiendo tiempos de respuesta ante una carga controlada.

5.4 Representación del Protocolo de Comunicación

El protocolo de comunicación corresponde a las capas medias de comunicación dentro de arquitecturas OSI [Open System Interconnection], y que comprenden básicamente a los niveles de red y transporte. La infraestructura de protocolos de comunicación utilizada en este estudio es TCP/IP.

Dentro del modelo, se requiere evaluar el tiempo de servicio de los paquetes en este protocolo. Ante un requerimiento por la red, el servidor responde al proceso cliente avisando su recepción. Registrando el tiempo en la máquina cliente al enviar el requerimiento, T_1, y al recibir la respuesta, T_2, se puede estimar el tiempo de servicio en el protocolo de comunicación. Con la carga de la red como entrada al FESC que la representa, se

obtiene el tiempo de transmisión, T_3, de un paquete. El tiempo del protocolo de comunicación, T_C, se estima como $((T_2-T_1)-T_3)/4$, dado que tanto el paquete de solicitud como el de respuesta "entra y sale" del protocolo.

5.5 Representación de la aplicación

La aplicación invocada puede representar en sí un modelo computacional relativamente complejo, que incluye, entre otros, contención por memoria, CPU y discos. Este es el caso de aplicaciones que proveen servicio de archivos o bases de datos. En general, la aplicación se puede representar por medio de un modelo abierto [9] (transaccional), representando cada componente crítico mediante centros de servicio, con una tasa de llegada que depende del número de clientes en acceso concurrente a la aplicación.

Considerando la provisión de "relojes" de control entre los distintas interfaces de comunicación, se pueden medir los tiempos de respuesta desde la aplicación, y compararlos con los tiempos que entrega el modelo. Dada la infraestructura cliente/servidor disponible, el proceso de medición se puede repetir en distintas máquinas servidoras, lo cual puede servir como un elemento adicional para la validación del modelo de la aplicación.

Cuando no sea posible medir exhaustivamente los centros de servicio de la aplicación, ésta se puede representar mediante una FESC, considerando el tiempo de respuesta global en función del número de clientes activos. La FESC debe representar variaciones de parámetros del equipo servidor (p.ej. velocidad de la CPU). Luego, es necesario cuantificar el retardo de la aplicación, en función del número de clientes concurrentes.

Para el proceso servidor, cada nueva solicitud es atendida por un subproceso que activa a la aplicación, y es independiente de la máquina a que responde. Por lo tanto, utilizar m máquinas cliente sólo aumenta la complejidad de las mediciones. Una ventaja de usar m máquinas cliente es aumentar la carga de la red para cuantificar la contención ante un aumento de tráfico, pero el contar con un generador de tráfico hace esto innecesario.

VI MEDICIONES PRELIMINARES

A continuación se presentan mediciones realizadas en el ambiente cliente/servidor descrito anteriormente. Estas mediciones pretenden mostrar parte de la metodología de evaluación de desempeño propuesta en este trabajo, y presentar una tendencia en el desempeño de distintas etapas involucradas en las transacciones cliente-servidor.

Se incluyó una máquina servidor (UNISYS U6050/UNIX V.3) y una máquina cliente (VAX 3400/ULTRIX-32 V3.1), usando como aplicación remota el sistema de base de datos SQLPLUS, y efectuando los clientes consultas del tipo "help select" (que entrega 6 Kbytes de información). En este caso no se incorporó carga artificial sobre la red, sino que sólo se varió el número de clientes concurrentes (de 1 a 10), percibiéndose una carga sobre la red que fluctuaba entre un 1% y un 3% de su capacidad. Los tiempos que se consideraron son:

- Conexión: tiempo que requiere el cliente para conectar su canal de comunicación con el servidor.
- Inicio aplicación: tiempo en que el proceso servidor crea directorios temporales, invoca a la aplicación, y crea un canal entre ésta y su salida (*output*).
- Aplicación: El tiempo que demora la aplicación para satisfacer el requerimiento y enviar la respuesta al cliente (este tiempo se deglosa en distintos niveles de desempeño en las transacciones cliente/servidor).

La fig. 4 muestra estos tres componentes de tiempo. Se aprecia que el "overhead" de iniciar una conexión remota es bajo comparado con el tiempo para satisfacer el requerimiento remoto. Al crecer el número de clientes, el tiempo de conexión aumenta levemente. El tiempo de inicio de la aplicación involucra la activación de un mayor número de procesos y la creación de áreas de trabajo temporales, y en este caso este tiempo es más sensible al aumento en el número de clientes.

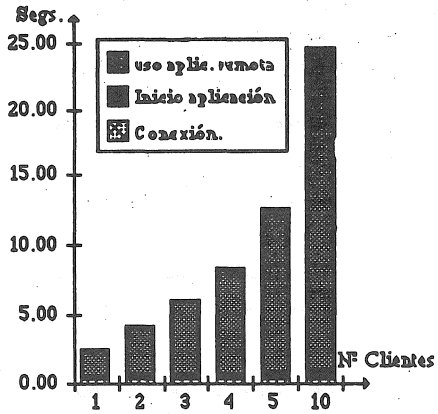


Figura 4: tiempos de respuesta

6.1 Deglose de tiempos en acceso a la aplicación remota

Dentro del tiempo de respuesta de la aplicación se incluye el tiempo que requieren los paquetes de datos en ser enviados desde la aplicación hacia el cliente. Aquí pueden ocurrir dos situaciones de retardo: espera del servidor por datos desde aplicación, y envío de los datos por la red y recepción de confirmación desde el cliente. Cuando hay espera de datos desde la aplicación, la red no produce contención.

Para estimar la proporción de tiempo en espera por la aplicación, se midió tanto el número de lecturas nulas desde la aplicación (al intentar leer desde la aplicación aún no hay datos) y el número de lecturas exitosas desde la aplicación (ver fig.5) Se puede apreciar que la congestión producida por la aplicación es alta.

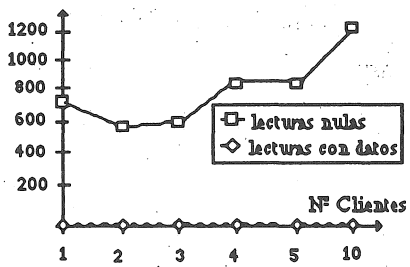


Figura5 Lecturas desde la aplicación

La fig. 6 muestra el tiempo de contención en la red. Se aprecia que este retardo es poco significativo dentro del tiempo total (ver fig. 4). El incremento en este tiempo es debido al aumento de "overhead" involucrado en el protocolo de comunicación (TCP/IP), dado que la carga en la red se mantuvo en un mínimo.

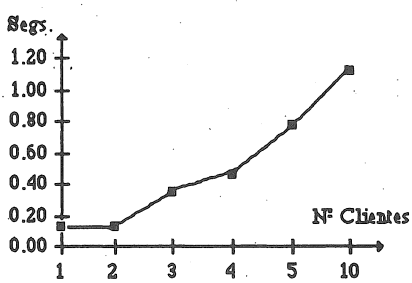


Figura 6. Contención en sistema de comunicación

Finalmente, en la fig. 7 se muestra la demanda de CPU requerida por cada proceso, en función de la demanda (en número de clientes). Cabe señalar que, por cada cliente existe un proceso en la máquina cliente y dos procesos en la máquina servidora. Esta demanda depende de la velocidad de la CPU tanto en el cliente como en el servidor.

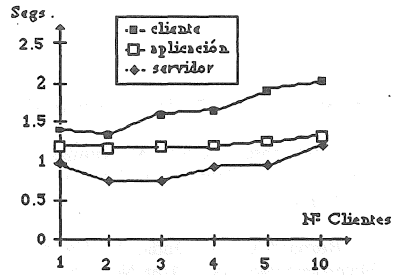


Figura7: Demandas de CPU por proceso

Mediciones como las anteriores se pueden realizar en distintos escenarios (tipos de computadores clientes y servidores, tráfico en la red, etc), y con diferentes aplicaciones remotas. Los datos identificados muestran distintos niveles de desempeño de aplicaciones cliente/servidor y se pueden incorporar a modelos de redes de espera como los descritos anteriormente.

VII. CONCLUSIONES

En este trabajo se propone una metodología para evaluar el impacto de los diferentes niveles de desempeño de una red, en aplicaciones cliente-servidor. Se realizaron herramientas de comunicación, para generar patrones de carga reales en una red y para medir el desempeño de los distintos elementos de interacción en las transacciones entre clientes y aplicaciones remotas.

Se ha definido un modelo general del sistema que, una vez llevada a cabo mediciones, bajo distintos escenarios de carga y configuraciones, permita predecir el impacto de los distintos niveles de comunicación, y definir cotas y patrones de desempeño en diversas aplicaciones de red.

Finalmente, se presenta un ejemplo de utilización de la infraestructura de evaluación desarrollada, que entrega resultados de desempeño en el acceso a una aplicación remota. A futuro, se espera crear una plataforma automatizada para la experimentación con la metodología de evaluación propuesta.

BIBLIOGRAFIA

[1] G. Almes and E. Lazowska. "The behavior of Ethernet-like Computer Communications Networks". Proceedings of the 7th Symposium on Operating Systems Principles, 66-81. 1979.

[2] F. Aurtenechea, I. Casas, M. Kutulas. "Diseño y Evaluación de un Servidor de Procesos para una Red Local". XVII Conferencia Latinoamericana de Informática", CLEI- Panel'91 Expodata, Caracas, Venezuela, 459-476 Julio 1991.

[3] G. Bernard G, A. Duda, Y. Haddad, G. Harrus. "Primitives for Distributed Computing in a Heterogeneous Local Area Network Environment", IEEE Transactions on Software Engineering, 1567-1578, Vol 15, No. 12, Dec 1989.

[4] P.I. Boulton, R.E. Soper, E.S Lee. "Simulation of Two LANs". Technical Report CSRI-233, University of Toronto. March 1990.

[5] W. Bux. "Performance issues in local-area networks". IBM Systems Journal, Vol 23, Nº 4, 351-374, 1984.

[6] D. Cheriton. "The V Distributed System". Communication of the ACM, Vol 31, No. 3, 314-333, March 1988.

[7] T. Gonsalvez. "Performance Characteristics of 2 Ethernets: an Experimental Study", ACM Sigmetrics, Performance Evaluation Review, Vol 13 Nº 2, August 1985.

[8] D. Jacobson, S. Gaitonde, J. Kim, J. Lee, D. Rover, M. Sarwar, M. Shafiq, "A Master/Slave Monitor Measurement Technique for an Operating Ethernet Network", IEEE Network Vol. 1, Nº 3, July 1987.

[9] E. Lazowska, J. Zahorjan, G.S. Graham, K.C. Sevcik, "Quantitative System Performance Computer System Analisys Using Queueing Network Models", Prentice Hall, New Jersey, 1984.

[10] E. Lazowska, J. Zahorjan, D. Cheriton, W. Zwaenepoel. "File Access Performance of Diskless Workstations". ACM Transactions on Computer Systems. Vol. 4 Nº 3, 238-268, August 1986.

[11] R. Metcalfe, R. Boggs "Ethernet: Distributed Packet Switching for Local Computer Networks". Communicaction of teh ACM, Vol. 19, Nº 7, 395-404, 1976.

[12] Quantitative System Performance Inc., "MAP - Release 1.2 User's Guide & Reference Guide", Seattle, Washington, 1985.

[13] K. Shah. "Managing networks of the '90s". Data Communications International, 95-106, December 1989.

[14] J.F. Shoch, J.A. Hupp. "Measured Performance of an Ethernet Local Network" CACM, 23:12, pp. 711-721, Dec. 1980.

[15] A. Tanenbaum. & R. Van Renesse. "Distributed Operating Systems". Computing Surveys, Vol. 17, Nº 4, 419-470, 1985.

CIG: a Class Interface Generator for C++

Ricardo Baeza-Yates *Mario Tichy*

Depto. de Ciencias de la Computación
Universidad de Chile
Casilla 2777, Santiago, Chile
E-mail: {rbaeza,mtichy}@dcc.uchile.cl *

Abstract

Development of an automatic interface generator for C++ classes, with graphical user interface (CIG), is reported. CIG is the outcome of a one semester workshop on object oriented software development for graduate students. The system is written in C++ and C, under the X-Windows environment. This paper gives an overview of the capabilities of the system and its design principles. Concluding remarks include comments on CIG uses and lessons learned from the project as a teaching experience.

Keywords: object oriented programming, C++, class hierarchy, graphical user interface, code generation, X-Windows.

1 Introduction

We feel that the nature of programming is changing. We do no longer think about our programs as individual projects where specific software is produced to meet some particular requirement. As applications become bigger and complex, there is a growing need to build on the work of others as well as on ours. For many years this was attempted through the use of function libraries. As we learned to get profit from the advantages of data abstraction, class libraries began to replace function libraries.

Under the scope of object oriented programming (OOP), software development consists in using objects from existing classes, that send messages one another; and defining new classes as needed.

Class construction involves two stages, that is: class interface definition, and class methods implementation. The later is a creative task that requires knowledge and reasoning. It is hard to automate. The former is a tedious one; often hours of code writing are needed to state a few synthetic ideas. So, it can be easily automated.

CIG was conceived as a programming tool allowing graphical specification of C++ class interfaces using icons, windows, menus, and dialog boxes with mouse support. Afterwards, the system checks consistency on programmer's request and automatically generates code.

We first describe the goals behind the project and the design principles used; then, the architecture of the system is described, and our main results are summarized in the last section.

2 Background

The project had two sets of goals: didactic and applicable. Applicable means that the need for a tool like CIG was rather evident. In the sequel the didactic goals are discussed.

Graduate students in our CS department are allowed to take a course on the theory of OOP. This course is complemented by a one semester workshop on object oriented software development. In

*This work was partially supported by Grant Fondecyt 91-0868 and Grant C-11001 of Fundación Andes.

the workshop, top-down design is taught by walking through designs. Reusability, maintainability and extensibility of programs is tested by trying to apply these concepts to actual code that solve real life problems. The principles of good design and programming are not presented as abstract sermons, but as examples in the context of concrete working programs.

A second concern of the workshop was to teach how to develop software in a team. The project was split into modules and each module was assigned to a group of two students. During the project, the groups were in charge of implementing their module and providing a friendly interface to the remaining parts. Sharing code was steadly encouraged to save time and effort. Extensibility and maintainability of the modules were tested by changes in the design specifications.

3 Design Principles

Design was one of the major challenges proposed in the workshop. A few general guidelines were given as specifications, but everything else was designed by the students. We stressed:

- Design by objectives: before proposing any solution to specific problems or planning any set of tasks, clearly state what are the purposes of the software that is going to be designed. Afterwards, take this in mind along all the design process. Clear objectives avoid wasting time and effort in useless work.

- Top-down design: every one seems to know what this buzz-word means; however, at the workshop we discovered that each one has a slightly different idea about it. Not withstanding this, an effective stratification of the design process was achieved according to different levels of abstraction.

- Object Identification: the question is not "how to find the objects in my problem?", but rather "what are the classes of objects with a common behavior in my problem?". There is no absolute answer which leads to a "software synthesis" technique. A successful selection of base classes requires talent and experience. A general approach and insight, can be found in Meyer [6, Section 14.2, Finding the Classes]. The approach called by Meyer "decomposition evaluation" became particularly fruitful. Since criticism is easier than design, we used a rough decomposition as a starting point. After a few revisions, good designs arisen.

- Building on software tools: great emphasis was put in avoiding to spend time and effort developing what already exists or doing by hand what can be automated. Reading [4] had been strongly recommended, and its philosophy was repeatedly used.

Following the recommendations in [6, Ch. 2] on modularity, CIG was split in a few modules regarding:

- modular decomposability - this is the most intuitive idea about modularity: decomposing a problem into several subproblems;

- modular structure - modules are build to be freely combined for the production new systems, possibly in an environment quite different from the current one;

- modular understandability - each module should be readable and understandable independently from other modules;

- modular continuity - a small change in the problem specification should lead to slight changes in just one or a few modules.

- modular protection - the effect of an abnormal condition in one module during execution, should remain confined to this module, or at least it should propagate only to few other modules.

Finally, it should be pointed that all this fine criteria were applied with a few restrictions regarding real world trade-offs, mainly arising from a limited availability of man-hours. This led, for instance, to grouping together software that otherwise would be split in more than one module, as will be better explained in the following sections.

4 Description of the System

CIG consists in four modules: the user interface, called GUI (for Graphical User Interface); the automatic code generator ACG; the input/output manager IOM; and the syntactic and semantic checker that also performs some additional tasks, arbitrarily included in the same module. Due to time constraints, the later was designated to perform as the communicator among the remaining modules; and, in a first prototype, it even replaced the GUI with a command driven interface. In the workshop, this module was named the Master, and so we will call it here. The interaction among the modules is depicted in Figure 1.

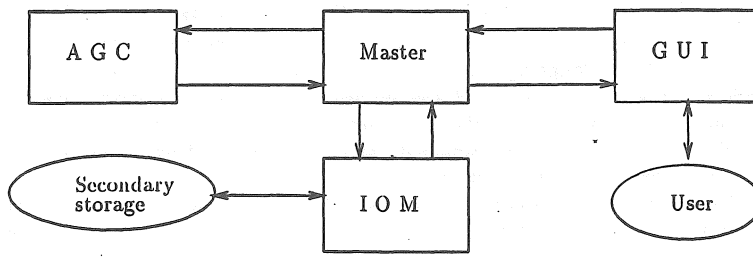


Figure 1: Schematic diagram for the four modules of CIG, and their interaction.

GUI was mainly written in ANSI C [5] and not in C++, to avoid compatibility problems with the code generated by the graphical design tools used under X-Windows. For this task the tool GUIDE (Graphical User Interface Development Environment) of Open Look [7] was used. GUIDE allows to "draw" interactively the interface, generating code that can be compiled under X-Windows using the XView Toolkit and the standard X libraries [8]. GUI is described in section 4.1.

The other three modules were coded in C++ [9], the Master was implemented with the aid of Lex and Yacc [2] (compiler construction tools available in Unix), IOM built on the standard libraries, and AGC was designed and coded as most text processing utilities.

At user's request, GUI tells Master which classes are in the hierarchy to be created, which instance variables they have, which are their methods, and what are the inheritance relationships among the classes. Besides, GUI may ask to perform a consistency checking for the proposed hierarchy, and one or several of the following tasks: storing a hierarchy in a file; deleting a hierarchy stored in a file; retrieving a hierarchy from a file; reading the names of the files in the current directory; changing the current directory; or generating code for a class or a set of classes.

Master passes the information received from GUI to AGC, one class at a time, and the later generates the code in an object of the class "File". This File can be passed to Master that routes it toward IOM which stores it in a class library. IOM is able to store in a file the code generated

within CIG, to retrieve old hierarchies for modifications, to update old hierarchies modified during a session, and to store new hierarchies created during the session. The following subsections describe each module.

4.1 The Graphical Interface

Since CIG was designed to ease the development of class libraries, the effectiveness of its user interface is an essential part of the system. The system provides several services that were assigned to different modules, but the distribution of tasks and the interface for most of these modules is unknown to GUI. The user interface only dialogs with the Master module which is in charge of all the internal interaction of the system.

Regarding the user, it is assumed that he/her is familiar with a window environment with mouse support such as X Windows, MS-Windows or the Macintosh user interface; that knows how to develop C++ class libraries, using conventional programming techniques; and that understands the representation of class hierarchies by means of a directed, acyclic graph. Familiarity with graphical user interfaces developed under Open Look is desirable, but is not a must.

It would be desirable a user interface that prevents or detects errors in the specifications given by the user. However, there are errors and inconsistencies hard to detect. Therefore, additional checkings must be performed by other module in the system. The later should work like no verification is provided by the user interface, and GUI should only take care of the errors that are easy to prevent and that may save time and effort in a later debugging process.

The main objective in designing GUI was to provide the user with a *simple* tool for expressing the structure and the behavior of the classes needed. Simple means a short learning time, with natural and intuitive interaction. Simplicity was seek by reducing the number of options provided. There is an obvious trade-off between simplicity and usefulness; we tried to include every essential option to the user, but not more, sacrificing completeness.

When CIG is invoked, it produces the opening of the main window, which is the parent for all the remaining windows. Edition is controlled by a set of menus available in the main window.

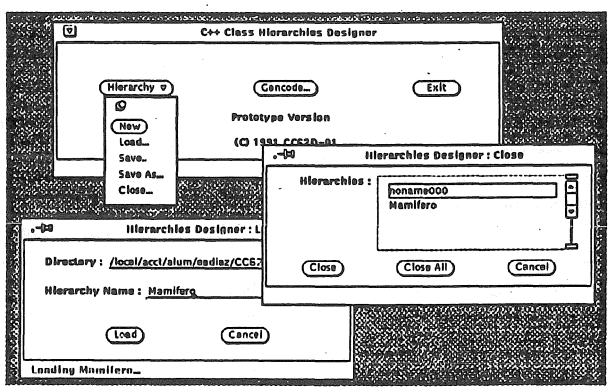


Figure 2: Main window.

4.1.1 The Main Window

This window has a bar with three buttons named (see Figure 2):

- Hierarchies: opens a second menu for hierarchy handling. Its options are:
  - New: opens a new empty edition window to create a new hierarchy.
  - Load: allows to load from a file an already existing hierarchy. The user is prompted to enter the hierarchy's name in a dialog box.
  - Save: allows to save to a file one, several, or all the hierarchies with which the user is working. The details are controlled with the aid of a dialog box.
  - Save as: allows to rename a hierarchy and to create a copy of a hierarchy with a different name.
  - Close: allows to close one, several or all the hierarchies with which the user is working. The names of the hierarchies are selected from a list.
- Gencode: this option gives a directive to generate code for one or several classes of a hierarchy.
- Exit: provides a quick way to close all the windows, saving all the new data and finishing the session.

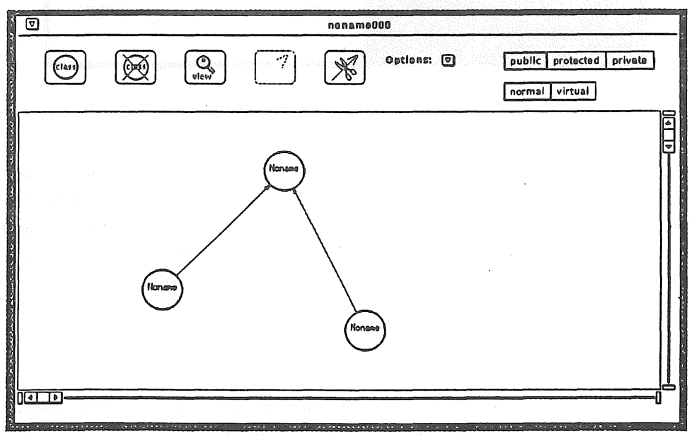


Figure 3: Edition window.

4.1.2 The Edition Window

This window has five icons that represent the available operations for dealing with the classes of a hierarchy. Besides these icons, two sets of exclusive buttons define the inheritance relation between the classes.

The icons (see Figure 3) allow the user to create a new class; destroy a class; open a dialogue box to declare instance variables and methods; create an inheritance arrow to link a pair of classes; and destroy an inheritance arrow.

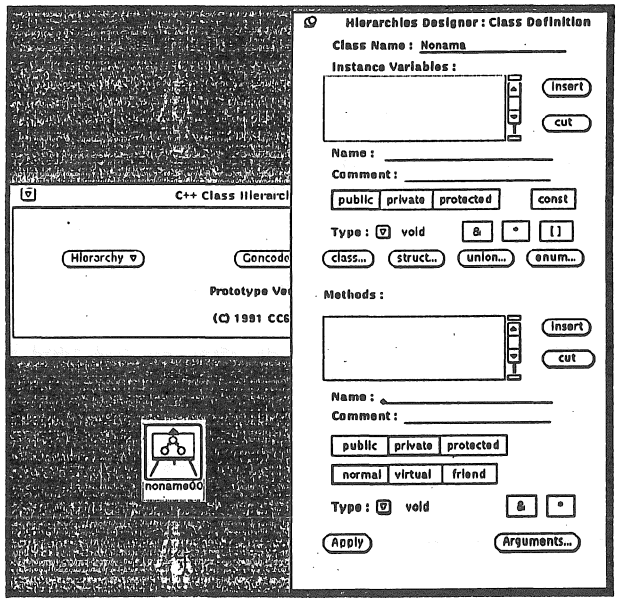


Figure 4: Class information.

A dialog box opens when clicking over the class icon (see Figure 4), where the following information about the class is available: the name of the class (every class, when created, is called "noname"); the names of the instance variables; a comment attached to each variable; the type of the variable; any legal qualifier; the section (public, private, protected) to which the variable belongs; and for each method and their arguments, similar information as that for the instance variables.

4.2 The Master Module

The main purpose of the Master module is to check consistency of user defined hierarchies. Ideally, it should be an object capable of receiving a "check" message and a pointer to a hierarchy, and should return a list of diagnosis messages. However, as already explained, the same module was put in charge of several additional tasks.

4.2.1 Consistency checking

A consistency checker should only verify that the coded information is complete and non contradictory. Some of the consistency rules are already implicit in the user interface. The consistency check includes several aspects, being some of them implemented in the first version of CIG. For example, consistent use of types; name uniqueness; potential problems in multiple inheritance; connectivity of the class hierarchy; etc. Most of the consistency problems are due to some peculiarities of the C++ language which are not usually well known by the user [3].

4.2.2 The additional tasks of "Master"

Besides checking consistency in the specifications given by user, the Master module dialogues with the ACG and IOM modules in order to:

- generate and store code for a class or set of classes;
- manage files and directories in order to store conveniently the information on class hierarchies, without needing to exit the application nor opening an additional window.

At present, GUI creates the remaining three modules of the system as unique objects of their class. In a future version of the system, it is planed to create first the Master, which will create GUI, ACG, and IOM objects besides a separate module exclusively in charge of consistency checking. Under this scheme, it would be possible to create more than a single object of each class (say, several identical modules) for assigning the tasks with more flexibility and running several processes.

4.3 The Automatic Code Generator

The automatic generation of C++ code is done by the ACG module. ACG receives from "Master" the specifications received from the user, one class at a time. This information is structured in such a way that ACG only needs to pass once over the data while it "fills on the blanks" to generate the code. Since CIG is intended only to create the interface for the classes, all the methods are declared "external", or the implementation is substituted by the comment: "Here should be placed the implementation." The user chooses one of these options.

4.4 The Input/Output Module

The interaction of CIG with the file system is done exclusively through IOM, the Input/Output module. IOM dialogues only with "Master", offering the following services:

- Storing all the information concerning one hierarchy in a single physical file.
- Deleting one or several files containing information on hierarchies. The files are selected with the mouse, from the list of files in the current directory.
- Retrieving a file with the information about a hierarchy. The file is selected with the mouse, from the list of files in the current directory.
- Storing the C++ code generated by ACG in a file of a specified directory.
- Listing all the files with information about class hierarchies which belong to the current directory, and listing all the subdirectories in the current directory.
- Warning when a request is going to cause the overwriting of an existing file, and asking if the old file should be copied before saving the new one.

5 Conclusions

The design and implementation of CIG in a workshop on object oriented software development allowed to draw several valuable conclusions from a didactic scope and from the analysis of the system value as a CASE tool.

Looking at the project as a teaching experience, the main conclusions are the following:

- A theoretical course on OOP is not enough to prepare graduate CS students to develop software with this innovative approach. This is true even when the course included homework and small projects on the practical aspects of the subject. We found that important concepts are misunderstood, the value of other concepts is neglected, and several central ideas are rejected due to misconcepts and prejudices.
- Almost all the students that took part in the workshop are not well prepared to work in a team. Usually they fail to explain their design decisions to members of the other groups, they avoid discussing about their own ideas by giving up without arguing or defending their points of view, and they refuse to establish deadlines for their work, rising several bottlenecks during the project.
- During the project, most of the students showed a remarkable positive evolution regarding all of the above mentioned problems. We believe that such courses may significatively contribute to the shaping of well trained engineers that will be able to participate in complex software projects.

Additionally, CIG has attracted the interest of many people involved in software development; it seems to really be a useful tool with a friendly user interface. However, at present, CIG is still a prototype. Some important features are not yet implemented or are implemented in an unsatisfactory form. Finally, the user interface is hardly portable since it was mainly developed with the aid of XView. The project included a Windows 3.0 version of GUI, which is one of the future research lines. The present state of CIG, besides being a very interesting teaching experience, should be regarded as an advanced prototype which may be used to test techniques of stepwise refinement, while the product itself deserves further efforts to reach a production level. For example, to include Booch's notation [1] in the graphical representation.

References

[1] Booch, G. Object Oriented Design with Applications, Benjamming Cummings, 1991.

[2] Brown, D. and Mason, T. Lex and Yacc, O'Reilly, 1990.

[3] Ellis, M. and Stroustrup, B. The Annotated C++ Reference Manual, Addison-Wesley, 1990.

[4] Kernighan, B. and Plauger, B. Software Tools in Pascal, Addison Wesley, 1976.

[5] Kernighan, B. and Ritchie, D. The C Programming Language, ANSI version, Prentice-Hall, 1988.

[6] Meyer, B. Object-oriented Software Construction, Prentice Hall Intl. (U.K.), 1988.

[7] Open Look: Graphical User Interface Application Style Guidelines, Sun Microsystems Inc., Addison-Wesley, 1990.

[8] Scheifler, R., Gettys, J. and Newman, R. The X-Window System, Digital Press, 1988.

[9] Stroustrup, B. The C++ Programming Language, second edition, Addison-Wesley, 1991.

GENERALIZACION DEL OPERADOR DE CRUCE EN ALGORITMOS GENETICOS

Barrios, D., Ríos, J., Segovia, J.
Facultad de Informática de Madrid (U.P.M.)
Departamento de Inteligencia Artificial
Campus de Montegancedo, 28660 Madrid

Abstract

Cuando se utilizan algoritmos genéticos en problemas de optimización, existen ciertas limitaciones respecto de la convergencia. En este trabajo se plantea una estrategia para evitarlas, consistente en la construcción de un nuevo operador, el Cruce Generalizado, que capacita al algoritmo para acceder a mayor diversidad de puntos. Para ello, se parte de propiedades extraídas del cruce ordinario. A partir de ellas, y dado el fundamento teórico que proporcionan, se efectúa la mencionada generalización.

1. Introducción

Con el método de optimización proporcionado por los Algoritmos Genéticos se parte de una muestra aleatoriamente elegida en el dominio de búsqueda, llamada población, que se transforma de generación en generación [4]. Cuando se llega a un estado estacionario, se entiende que la población está constituida por las soluciones proporcionadas por el algoritmo. La transformación se realiza con la aplicación de diversos operadores, y el algoritmo actúa sobre una codificación del espacio de búsqueda.

La relación que existe entre progenitores y descendientes, mediante los operadores de cruce conocidos, tiene como consecuencia restricciones sobre los puntos a obtener. Por otra parte, el teorema de Holland [6] enuncia que la probabilidad de cada esquema de sobrevivir al cruce aumenta cuando la longitud y el orden de éste, con las definiciones habituales, disminuye. Si bien el orden no presenta problemas, ya que se puede pensar en la solución como intersección de esquemas de orden 1, la condición

sobre la longitud resulta indeseable.

La solución que se propone en este trabajo consiste en la transformación del cruce, de modo que éste sea capaz de efectuar una búsqueda más exhaustiva, al tiempo que se logre una independencia absoluta entre la longitud de los patrones y su capacidad de sobrevivir.

2. Propiedades del cruce ordinario

Las propiedades y definiciones que se van a estudiar en esta sección establecen las bases necesarias para la generalización del operador de cruce.

Propiedad 2.1:

Sean x_1 y x_2 son dos elementos cualesquiera del espacio de búsqueda. Sean x'_1, x'_2 el resultado del cruce genético de los elementos anteriores. Entonces se tiene que:

$$x_1+x_2=x'_1+x'_2$$

En lo que sigue, x_1, x_2 son dos elementos del espacio de búsqueda, de representación binaria $a_0a_1...a_n$ y $b_0b_1...b_n$ respectivamente. La representación decimal de dichos elementos será $a_0+a_1.2+...+a_n.2^n$ y $b_0+b_1.2+...+b_n.2^n$ respectivamente.

Definición 2.1:

*Se llama "x_1 ó x_2" al punto que resulta como consecuencia de aplicar la función lógica **or** a las respectivas codificaciones de x_1 y x_2. Dicho elemento de escribirá como:*

$$x_1 \vee x_2$$

y su expresión en binario será $d_0d_1...d_n$, siendo d_i=Máx $\{a_i, b_i\}$. Se llama "cadena máxima" de x_1 y x_2 a esta última cadena.

Definición 2.2:

*Se define "x_1 y x_2" como el punto que resulta de aplicar la función lógica **and** a las respectivas codificaciones de x_1 y x_2. Dicho elemento se escribirá como:*

$$x_1 \wedge x_2$$

y su expresión en binario será $d_0d_1...d_n$, siendo d_i=Mín $\{a_i, b_i\}$. Se llama "cadena mínima" de x_1 y x_2 a esta última cadena.

Propiedad 2.2:

Dados los puntos x_1 y x_2 del espacio de búsqueda,suponiendo que se cruzan y que el resultado del cruce son los puntos x'_1 y x'_2 ,se tiene:

$$i)\quad x_1 \wedge x_2 \leq x_1, x_2 \leq x_1 \vee x_2$$
$$ii)\quad x_1 \wedge x_2 \leq x'_1, x'_2 \leq x_1 \vee x_2$$

Es decir, tanto los los puntos que se cruzan, como los descendientes de éstos mediante dicho cruce, están limitados al mismo intervalo, llamado en adelante *"intervalo de cruce"*, determinado por los extremos correspondientes a las definiciones 2.1 y 2.2.

3. Definición y análisis del operador de Cruce Generalizado

Con el nuevo operador de cruce que se propone en este trabajo, llamado *Cruce Generalizado* en adelante, se podrá acceder a cualquier punto del intervalo de cruce, y únicamente a ellos. Para ello, si dos elementos cualesquiera de la población resultan elegidos para la aplicación del operador, el resultado serán los elementos x'_1 y x'_2 tales que:

$$x'_1 \in [x_1 \wedge x_2,\ x_1 \vee x_2]\ ,\ x'_2 = (x_1 + x_2) - x'_1$$

siendo x'_1 un punto, elegido aleatoriamente en el intervalo, siempre que los valores alcanzados por la función objetivo en éstos puntos supere, o al menos iguale, la que se tenía antes de hallar x'_1 y x'_2. En caso contrario, se tomará como producto del cruce aquellos dos puntos, elegidos entre los progenitores y los descendientes, cuyas imágenes sean las mayores.

En el siguiente lema se prueba que el resultado de aplicar el nuevo operador a dos puntos se mantiene dentro del intervalo de cruce.

Lema 3.1:

Sea $y=(x_1+x_2)-x$, tal que:

$$x \in [x_1 \wedge x_2 , x_1 \vee x_2]$$

Entonces se tiene:

$$y \in [x_1 \wedge x_2 , x_1 \vee x_2]$$

Demostración:

Basta tener en cuenta la propiedad:

$$x_1 + x_2 - (x_1 \wedge x_2) = x_1 \vee x_2$$

ya que,por ser $y=(x_1+x_2)-x$, se tienen las desigualdades siguientes:

$$(x_1 + x_2) - (x_1 \vee x_2) \leq y \leq (x_1 + x_2) - (x_1 \wedge x_2)$$

de donde:

$$x_1 \wedge x_2 \leq y \leq x_1 \vee x_2$$

El siguiente resultado formaliza la idea de que el cruce generalizado conserva el concepto de "herencia" [2], transmitida en forma de parte común entre cadenas, y establecido el otros operadores de cruce. El significado de esto es que, si bien es posible que los descendientes se alejen entre si, existe cierto control entre la distancia que guardan.

Supóngase cierta generación formada por los puntos $x_1, x_2, \ldots x_p$ del espacio de búsqueda. Supóngase elegidos dos puntos, x_1 y x_2 sin pérdida de generalidad, para el cruce generalizado. Sea I el intervalo de cruce para estos puntos. Si las cadenas correspondientes a x_1 y x_2 son respectivamente:

$$x^1{}_1 x^1{}_2 \ldots x^1{}_m x^1{}_{m+1} \ldots x^1{}_n$$
$$x^2{}_1 x^2{}_2 \ldots x^2{}_m x^2{}_{m+1} \ldots x^2{}_n$$

y se tiene:

$$x^1{}_i = x^2{}_i , \forall i \geq m+1$$

siendo m el máximo entero en estas condiciones, entonces la longitud del intervalo I está dada por:

$$\rho(I) = \sum_{x^1{}_i \neq x^2{}_i,\ i \leq m} 2^i$$

Lema 3.2:

$$\rho(I) < 2^{m+1} \Leftrightarrow \forall y \in [x_1 \wedge x_2\ ,\ x_1 \vee x_2]\ ,\ si\ y = \sum_{i=1}^{l} y_i 2^{i-1}\ ,\ y_i \in \{0,1\},$$

$$se\ tiene\ y_i = x^1{}_i = x^2{}_i\ ,\ \forall i \geq m+1$$

Demostración:

i) Suponiendo la longitud del intervalo acotada por 2^{m+1}, por poderse expresar dicha longitud como

$$\rho(I) = \sum_{\substack{x^1{}_i \neq x^2{}_i \\ i=1,\ldots,n}} 2^i < 2^{m+1}$$

se tiene que

$$x^1{}_i \neq x^2{}_i, \forall i \geq m+1$$

Por tanto, cualquier punto y contenido en el intervalo, si se expresa en la forma

$$y = \sum_{i=1}^{n} y_i 2^{i-1},\ y_i \in \{0,1\}$$

como debe ser

$$Min\{x^1{}_i, x^2{}_i\} \leq y_i \leq Max\{x^1{}_i, x^2{}_i\}, \forall i$$

se tiene que

$$y_i = x^1{}_i = x^2{}_i, \forall i \geq m+1$$

ii) Recíprocamente, si existe al menos un punto y en el intervalo que verifica

$$y_i = x^1{}_i = x^2{}_i$$

para todo i≥m+1, se tiene que la cadena mínima y la cadena máxima, para x_1 y x_2, coincide en las posiciones correspondientes a i≥m+1. Por tanto, la longitud del intervalo será

$$\rho(I) = \sum_{\substack{x^1{}_i \neq x^2{}_i \\ i \leq m}} 2^i$$

evidentemente acotado por 2^{m+1}.

4. Resultados de orden práctico

En general quedó probada una mejor convergencia usando el nuevo operador. Incluso con ejemplos donde existen dificultades para los algoritmos genéticos convencionales, con el nuevo operador de cruce mejora la convergencia. En otros casos, en los que se probó funciones que conducen a problemas paradójicos con el operador ordinario, se llegó al resultado deseado con el cruce generalizado.

En todos los casos, se utilizó codificación en binario, correspondiendo cada cadena de longitud 10 a la expresión en base dos del punto correspondiente, y siendo el intervalo de búsqueda el [0 , 1023]. Todos los parámetros, que aparecen en la tabla 1, se mantuvieron constantes durante los experimentos.

Tamaño de población	Probab. cruce	Probab. mutación
50	.5	.01

-Tabla 1: Parámetros-

En la tabla 2, la segunda columna corresponde al óptimo buscado, la tercera al alcanzado usando cruce generalizado, y la última al alcanzado con cruce ordinario. En las dos últimas columnas se indica, entre paréntesis, el número de generaciones de convergencia en cada caso.

$f_1(x)=(128/3)(x-500)$, $500< x <512$ $f_1=100$ en el resto.	512	512 (4)	781 (20)
$f_2(x)=x$	1023	1023 (9)	944 (19)
$f_3(x)=(x-1023)^2$	0	0 (9)	2 (20)
$f_4(x)=1/(x+1)$	0	0 (5)	4 (4)
$f_5(x)=x$, $x < 1023/2$ $f_5(x)=-(x-1023)$,resto	511	511 (10)	404 (8)

-Tabla 2: Funciones test-

5. Conclusiones

Se ha definido un operador de cruce, con el cual es posible una búsqueda más exhaustiva en todo el dominio, resolviendo problemas de convergencia subóptima que, de otro modo, pueden presentarse, tanto debido a problemas paradójicos, como a escaso tamaño de la población o a otras dificultades intrínsecas de cada problema.

Con el cruce generalizado se produce una mayor diversidad de población en cada generación, con lo cual se evita la convergencia prematura. El teorema de Holland se modifica, resultando independiente la probabilidad de supervivencia de cada patrón de su longitud. Por tanto, se obtiene una base teórica que permite deducir la mejor convergencia cuando se utiliza este operador.

Referencias

[1] J. D. Bagley, The behavior of adaptative systems which employ genetic and correlation algorithms, Doctoral dissertation, University of Michigan (1976).

[2] J. E. Baker, Adaptive selection methods for genetic algorithms, Proceeding of an International Conference on Genetic Algorithms and Their Applications, 101-111,

(1985).

[3] K. A. De Jong, Analysis of Behavior of a Class of Genetic Adaptive Systems, Ph. D. dissertation, University of Michigan (1975).

[4] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning (Addison-Wesley Publishing Company, Inc. 1989)

[5] J. H. Holland, Adaptation in natural and artificial systems (The University of Michigan Press, 1975).

[6] G. Sywerda, Uniform Crossever in Genetic Algorithms, submited to the 1989 International Conference on Genetic Algorithms.

[7] D. Whitley, Using Reproductive Evaluation to Improve Genetic Search and Heuristic Discovery, Genetic Algorithms and their Applications: Proceeding of the Second International Conference on Genetic Algorithms (1987) 108-115.

[8] D. Barrios, Operador de Cruce Generalizado en Algoritmos Genéticos, Tesis Doctoral. Octubre 1991.

[9] D. Barrios, A. Pazos, J. Ríos, J. Segovia, Oscillanting Problems in Genetics Algorithms, enviado y recibido por Complex Systems, USA.

Métodos para el cálculo de la Función Descriptiva en sistemas dinámicos no lineales

Gerónimo Bellassai, Luciano Ferrari

Laboratorio de Electrónica Digital
Universidad Católica
"Nuestra Señora de la Asunción"
Po. Box 1718
Asunción - Paraguay
email: lferrari@ledip.py

Un capítulo importante de la teoría de sistemas es, sin duda, la dedicada a sistemas dinámicos no lineales. Este estudio es de gran interés, sobre todo en el modelado de sistemas y resolución de problemas de control.
Se puede decir que las problemáticas ligadas al análisis de sistemas de control no lineales hacen parte de un conjunto de problemas todavía abiertos. Sin embargo, no existe hasta ahora una teoría sólida en grado de dar respuestas exactas a todos los aspectos de estas problemáticas. El método de la Función Descriptiva (FD) se coloca en este contexto, y en particular constituye una herramienta aproximada, de tipo heurístico, para la búsqueda de oscilaciones permanentes. El trabajo hecho tiene como objetivo encontrar de manera rápida y eficiente la FD como primer paso para la implementación de programas de simulación relativos a un sistema hecho.

1. Introducción

Informalmente, la Función Descriptiva (FD) se puede ver como una especie de función de transferencia para sistemas no lineales. Este método permite entender cuales son los factores que determinan, por ejemplo, la existencia y la estabilidad de las oscilaciones en los sistemas realimentados; la importancia y los límites de sus interconexiones.
La FD entonces puede ser de gran importancia para la comprensión de importantes fenómenos no lineales de los sistemas de control y en la síntesis de controladores no lineales.

No existen métodos generales y directos para el cálculo de la FD a partir de las ecuaciones de estado y de salida de un determinado sistema. La única metodología existente está basada en el concepto de cálculo numérico de la respuesta periódica del sistema solicitado con un ingreso sinusoidal.

En el caso de que la respuesta sea única y asintóticamente estable, el método más simple para el cálculo (en un punto) de la FD se puede describir como sigue:

1. Simulación del sistema, con un adecuado algoritmo de integración numérica, a partir de condiciones iniciales arbitrarias, bajo la acción de un ingreso sinusoidal de amplitud y frecuencias conocidas:

$$u(t) = U \sin(\Omega t)$$

2. Espera de la correspondiente condición de régimen permanente (periódico).
3. Cálculo del primer armónico de la respuesta periódica :

$$y(t) = Y \sin(\Omega t + \alpha)$$

4. Cálculo de la FD en función de (U,Ω):

$$D(U,\Omega) = \frac{Y}{U} e^{j\alpha}$$

Volviendo a repetir el procedimiento para distintos valores de U y Ω, se optiene la representación por puntos de la FD. Se puede proceder con posteriores análisis por medio de técnicas de interpolaciones (polinomiales, spline cúbicas, etc).

En este esquema, la segunda fase es la más engorrosa: la espera "espontánea" de un régimen periódico. El esfuerzo de este trabajo es el de reducir el tiempo de llegada del sistema al régimen permanente.

Definición del problema implementado.

El sistema dinámico no lineal tomado en consideración es el que describe el movimiento de un péndulo simple:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = - k \sin(x_1) - h\, x_2 + Amp \sin(\omega t)$$

Medios computacionales.

El trabajo se ha desarrollado en ambiente Apple/Vax. Se utilizó a tal fin, las funciones de integración de la aplicación Mathematica de Wolfran Resource Inc Dicha aplicación tiene un "front end" en el ambiente Macintosh y el "kernel" de cálculo en ambiente Vax.
El tipo de ambiente de cálculo numérico usado no es esencial, ya que solo se quiere demostrar la viabilidad de algunas hipótesis que a posteriori se podrían implementar con herramientas mas poderosas.

2. Solución Banal

Introducción

En el caso considerado de un péndulo simple, se logra el régimen después de muchas decenas de períodos con tiempos de cálculo del orden de la hora si al sistema, a partir de una condición inicial arbitraria, se lo deja evolucionar hasta su estado de oscilación permanente.
El problema, entonces, no está en encontrar nuevos medios de cálculo o más eficientes algoritmos de integración numérica, sino en individualizar métodos computacionalmente eficientes para el cálculo de la respuesta periódica. De este modo, sistemas intrínsecamente lentos pueden, a veces, evolucionar con trayectorias distintas de las "naturales".
Este tipo de metodología es, de todos modos, necesaria para tener un punto de referencia.

Definiciones teóricas

El método "banal" consiste en definir primeramente determinadas condiciones iniciales relativas al tiempo, a la posición angular y velocidad angular (X_1,X_2). En segundo lugar, se integra repetidamente la ecuación del péndulo real representada por el sistema, hasta alcanzar el régimen periódico. La primera interrogante que surge se refiere a la condición de régimen, es decir, ¿cuándo se puede afirmar haber llegado a la condición final?
Para contestar a esta pregunta tenemos la necesidad de hacer referencia a la definición de "norma" de un vector $\underline{X}$.

Definición 2.1: Norma esférica

$$\|\overline{x}\|_2 = \sqrt{(\overline{x}, \overline{x})} = \sqrt{\sum_{i=1}^{n} \overline{x}_i^2}$$

Definición 2.2: Distancia entre dos estados

La distancia entre dos estados la podemos entonces definir como la norma de la diferencia de los respectivos vectores.

Definición 2.3: Condición de régimen

El estado inicial se considera igual al estado final obtenido después de un periodo, cuando la distancia entres los dos estado es tal que:

$$\|\overline{x}(T) - \overline{x}(0)\| \leq \varepsilon$$

En la práctica, considerando que se dispone del resultado de la integración expresada puntualmente, el valor ε tiene que ser una fracción de la máxima distancia entre dos puntos cualquiera del movimiento de estados durante el período T. Una idea gráfica de esta definición podría ser la de figura 2.1.

3. Método de reinicialización mediante aproximación lineal

Introducción

La primera solución al problema definido arriba se denomina " *Método de reinicialización mediante aproximación lineal*". Este método, bajo la acción del ingreso sinusoidal, preve la búsqueda de una condición inicial que permita al estado del sistema llegar, en un período, a la condición de régimen periódico, es decir, cumplir una trayectoria cerrada. La idea central es calcular antes de cada nueva integración, la condición inicial aprovechando toda la información contenida en la última iteración. Para ello, se necesita linealizar el sistema en el entorno de duración de un período y se usará la matriz de monodromía para determinar la nueva condición inicial. Se repite el procedimiento hasta llegar a la condición de régimen .

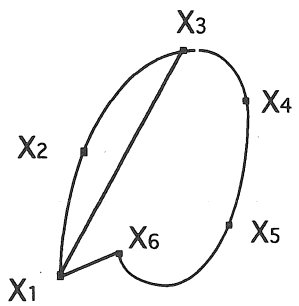


Figura 2.1

Tratado teórico

Introducimos unos conceptos para la comprensión de lo que se trata a continuación.
Dado un sistema lineal descripto para las ecuaciones:

$$\dot{x}(t) = F(t)\, x(t) + G(t)\, u(t)$$
$$y(t) = H(t)\, x(t)$$

La fórmula de Lagrange permite el cálculo del movimiento del sistema arriba en el caso sea lineal, regular y a dimensiones finitas:

$$x(t) = \Phi(t)\, x(t_0) + \int_{t_0}^{t} \Phi(t,\xi)\, G(\xi)\, U(\xi)\, d\xi$$

donde t_0 es el instante inicial.

Es claro que este cálculo necesita el conocimiento de $\Phi(t,t_0)$ nota como la matriz de transición o de monodromía. Cada columna contiene los movimientos libres relativos a condiciones iniciales particulares .
Normalmente la fórmula de Lagrange permite el cálculo del movimiento del sistema solo en los casos en los cuales la matriz de transición se puede determinar por la integración de la ecuación diferencial:

$$\frac{d}{dt}\Phi(t,t_0) = F(t)\,\Phi(t,t_0) \qquad \Phi(t,t_0) = I$$

Para los sistemas lineales invariables en particular, la matriz de transición es:

$$\Phi(t,t_0) = E^{F(t-t_0)} \qquad 3.1$$

Entonces la fórmula de Lagrange poniendo por simplicidad $t_0 = 0$ es

$$x(t) = E^{Ft}\,x(0) + \int_0^t E^{F(t-\xi)}\,G(\xi)\,U(\xi)\,d\xi$$

El objetivo, suponiendo conocer la función descriptiva del sistema linealizado alrededor del movimiento optenido de la precedente integración, será el de imponer que el sistema alcance la condición de régimen en un solo período T.
Considerando la función lineal en trozos, es posible determinar con la 3.1 la matriz de transición en cada trozo:

$$\Phi(t_n,t_{n-1}) = E^{F(t_n-t_{n-1})}$$

utilizando una propiedad de las matrices de transiciones que deriva desde la linealidad:

$$\Phi(t_n,t_{n-2}) = \Phi(t_n,t_{n-1})\,\Phi(t_n,t_{n-2})$$

se obtiene la matriz de transición sobre todo el período: $\Phi(T,0)$.

En la práctica, con el procedimiento descripto, disponiendo de una matriz de estado F por cada valor tabular del estado, se ha integrado un sistema lineal de tiempo variante obtenido por la linealización del original.

Por este sistema lineal, solicitado al mismo ingreso $u(\cdot)$, la fórmula de Lagrange es:

$$\delta x(t) = \Phi(t,t_0)\,\delta x(t_0)$$

La condición inicial que lleva a régimen el sistema linealizado en un solo período se obtiene imponiendo:

$$x(0) + \delta x(0) = x(T) + \Phi(T,0)\,\delta x(0)$$

donde:

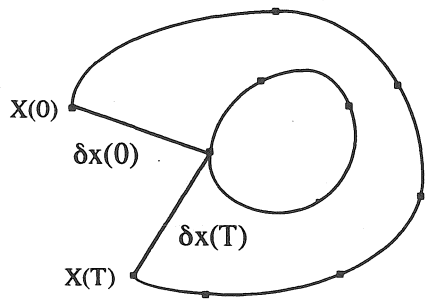
$x(0)$	es el valor del estado inicial
$\delta x(0)$	es la variable incógnita, es decir el incremento del estado inicial que determina la nueva inicialización.
$x(T)$	es el valor del estado final resultado por la última integración.

$\Phi(T,0)\ \delta x(0)$ es la variación del estado después de un período T

entonces el nuevo estado con el cual reinicializar el sistema para ejecutar la integración sucesiva es:

$$\delta x(0) = (x(T) - x(0))\ (I - \Phi(T,0))^{-1}$$

para dar una idea gráfica:



4. Herramienta utilizada

Como ya anticipado se ha utilizado el paquete software Mathematica. El tipo de herramienta de tal aplicativo resulta suficiente para tal objetivos.
Para la integración del sistema se ha usado la función RungeKutta que implementa el homónimo método para la solución de ecuaciones diferenciales. La función permite de variar el paso de integración en función de la precisión requerida.
La función ExpMatrix se ha revelado muy eficiente para el calculo de E^F donde F es la derivada con respecto al estado de la matriz de estado del sistema original.
Mathematica ademas de tener todas las características de un lenguaje de alto nivel, dispone de un conjunto de funciones de graficación muy útiles sobre todo en fase de análisis de los resultados.

5. Análisis de los resultados

El campo de variación de los dos parámetros de Amplitud y Frecuencia Angular de la forzante sinusoidal, se han elegido después de unas pruebas y en función de mostrar los casos mas significativos.
En particular para la Amplitud se han utilizado los valores: {1, 2, 4, 8}
y para la Frecuencia Angular:{ 0.1, 1, 5, 10, 100}.
Dado que el objetivo es de reducir el tiempo de cómputo para llegar a la condición de oscilación permanente, los parámetros observados son el número de periodos y el tiempo de cómputo necesario para lograr tal estado.
Todos los datos encontrados se han incluido en la siguientes tablas:

Frec \ Amp	1	2	4	8
0.1	P= 5 Time= 1337	P= 4 Time= 1374	P= 5 Time= 1543	P= - Time= NoEst
1	P= 18 Time= 409	P= 16 Time= 406	P= 16 Time= 546	P= - Time= NoEst
5	P= 85 Time= 664	P= 87 Time= 797	P= 82 Time= 838	P= 85 Time= 952
10	P= 141 Time= 823	P= 133 Time= 883	P= 138 Time= 933	P= 133 Time= 1014
100	P= 446 Time= 1464	P= 1 Time= 3.5	P= 1 Time= 4.8	P= 1 Time= 5.7

Solución Vanal

Frec \ Amp	1	2	4	8
0.1	P= 3 Time= 914	P= 3 Time= 1086	P= 3 Time= 1358	P= - Time= NoEst
1	P= 2 Time= 58	P= 3 Time= 104	P= 3 Time= 161	P= - Time= NoEst
5	P= 2 Time= 18	P= 2 Time= 21	P= 2 Time= 23	P= 3 Time= 41
10	P= 2 Time= 13	P= 2 Time= 15	P= 2 Time= 15	P= 2 Time= 17
100	P= 2 Time= 75	P= 1 Time= 3.5	P= 1 Time= 4.9	P= 1 Time= 5.6

Método de reinicialización mediante aproximación lineal

Confrontando los resultados de las dos tablas se pueden notar como el método de reinicialización es sin duda mucho mas eficiente de la solución banal.
Algunos datos son todavía anómalos. Las duplas Amplitud - Frecuencia Angular: <8 , 0.1>, <8 , 1> dan resultados que son afectados por el tipo de sistema utilizado. Si la variable X_1 supera el valor de $\Pi/2$ el sistema se vuelve instable y no se pueden lograr ningún punto de oscilación permanente. A nivel intuitivo este es el caso de una forzante que lleva el péndulo a una variación tal que lo obliga a volver atrás.

Los otros datos anómalos se refieren a la ultima fila de resultados donde el movimiento forzante tiene una frecuencia angular mucho mas elevada con respecto a la frecuencia angular natural del sistema cuyo valor se encuentra al rededor de 3 radiantes por segundo. Por lo tanto en un intervalo de integración de un periodo la onda perturbadora no es prácticamente influenciada por el movimiento natural del sistema. Nuestra condición de detección de estado de oscilación permanente releva una solución supuestamente de llegada.

El tiempo total de computo en la región de datos consistentes, Amplitud = {1, 2, 4} y Frecuencia Angular {0.1, 1, 5, 10} es entonces:

Solución Banal	10.553 segundos
Método de reinicialización mediante aproximación lineal	3.786 segundos

6. Conclusión y pasos sucesivos

El método denominado de la "reinicialización mediante aproximación lineal" resultó sorprendentemente eficaz respecto al método banal, reduciendo sensiblemente el tiempo de cómputo. Se concluye entonces que ésta puede ser una vía eficiente para el cálculo de la Función Descriptiva y la siguiente fase de simulación de sistemas no lineales.

Tres son los pasos que se quieren enfrentar en la línea del trabajo realizado:

1. Lo valores obtenidos demuestran que existe una relación entre todos los valores con igual frecuencia angular. Es pensable una mejoría de las performance si se inicializan la computaciones a partir de resultados de régimen ya optenidos?

2. Una segunda hipótesis que se quiere confrontar con las precedentes, es la búsqueda, en el conjunto de movimientos periódicos, de la solución del sistema; siempre bajo la acción de un ingreso sinusoidal. Se quiere llegar iterativamente a la solución a partir de un movimiento cualquiera, que sea periódico, aportando correcciones sucesivas con el fin de reducir el error.

3. Una vez comprobada la hipótesis más eficiente, se utilizarán otros tipos de herramientas computacionales para analizar la posibilidad de reducir el tiempo de llegada a las condiciones de régimen permanente.

Bibligrafia

L.A. Zadeh, C.A. Desoer *Linear System Theory* Mc Graw-Hill , 1963
R.E. Kalman, P.L. Falb *Topics in Mathematical System Theory* Mc Graw-Hill , 1969
S. Rinaldi *Teoria dei Sistemi* Clup 1977
C.L. Phillips, R.D. Harbor *Feedback Control Systems* Prentice-Hall 1991
B.C. Kuo *Sistemas de Controle Automaticos* Prentice-Hall 1982
S. Wolfram *Mathematica* Addison Wesley 1988

Stack Structure for the efficient execution of the FORTH Language Instruction Set

Alvaro Bernal N.

Abstract

This paper presents an alternative design approach of a stack structure for the efficient execution of high level language instructions. The stack system was designed to execute a FORTH directed instruction set. The instruction set to be executed, the structure of a basic cell and the control logic are described. A 32 bit word length prototype, with 12 ns access time, using an 1.2 μm double level metal CMOS technology, was designed.

I. Introduction

In most applications, data are conveniently organized in a stack structure. Usually, the stack is implemented through a set of registers whose contents are managed on a last-in-first-out (LIFO) basis. A stack is commonly used with arithmetic and logic operations, comparison blocks and for saving the return address for subprogram calls. Also, it facilitates the evaluation of expressions and minimizes the control overhead needed to store data [1].

The processors with oriented stack architecture, also called stack machines, use a stack extensively for temporary data storage and execution of all operations on the data stored there. This extensive use of stacks leads to the creation of efficient high-level language compilers [2], resulting in very efficient machine code execution. The processor operational units find all the data in a predefined location which is implied by the operation code itself, and no explicit address is required to be decoded, resulting in faster instruction execution.

Forth is a structured programming language [3], where most operations communicate only through a stack. This language uses a two stack programming model: the operand stack and a control flow stack. In this job, we present the stack structure used in a FORTH directed microprocessor with stack-based architecture being designed in the Integrated Systems Laboratory of São Paulo University.

The cell used in the operand stack design, leads to a structure which permits both a sequential addressing, like as RAM structure and to shift the data stored into the stack, like as shift register bank, allowing to execute some composite Forth instructions in one instruction cycle, so, this fact represents the main characteristic of this configuration.

II. The stack design approaches

The FORTH two stack programming model is supported by two stack caches in the data path. Both operand and return stacks consist of thirty-two words of 32 bits register. Most Forth primitives take operands from one or both stacks, push or pop the stacks and return a result to one of the stacks.

There are two common ways to implement stacks, first, using a shift registers bank and second, like a RAM structure.

A. Shift Register Approach

Lets consider a typical shift register approach. The stack is implemented using a n-array of shift registers (SR) $S_1,....S_n$, and only four control signals must be generated for each bit cell. The control signals allow to execute three basic operations:

PUSH: (S_{i+1} <------ S_i , $1 < i < n-1$)
POP : (S_i <------- S_{i+1} , $1 < i < n-1$)
NOP : (S_i <------- S_i , $1 < i < n$)

In a LIFO structure, this configuration allows to push and pop information only for the top cell. In case of a FIFO structure, the information can be pushed on the top cell, and poped on the bottom cell. The main disadvantage of this configuration consists of the high power consumption when a POP or PUSH operation is executed, due to the fact that all previous stored data must be shifted. See figure 1 [4].

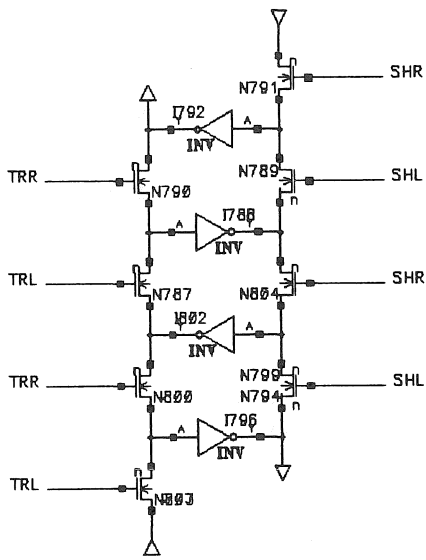


Figure 1. Array of shift register

B. RAM approach

Another possibility to implement a stack, is like a memory structure. Only few pointers are needed to indicate the actual top address. The stack pointer contains the address of the most recent stored item. The operations PUSH and POP would be executed as:

PUSH:
S_i <-- S_{i+1}, $1 < i < n$
RET[S_i] <-- SP

POP:
SP <--- RET[S_i]
S_i <--- S_{i-1}, $1 < i < n$

In the shift register approach, the pointer always is addressing the same top position and the data are shifted down or up. In the RAM approach, the data are stored in fix positions and the access to any location can be realized through the variable pointer. This configuration allows to read the last stored item but does not allow to shift data into the structure. Furthermore, due to the fact that the addressing is sequential, not is allowed to have random data access from the structure.

III. Stack Structure

Our stack system has been designed mixing the most interesting concepts from both shift register and RAM approaches. So, the operand stack allows partial random access and the return stack is a pure LIFO. Figure 2 shows the block diagram of the operand stack, bellow we describe them:

A. Return Stack

The return stack was implemented as a pure LIFO structure. The memory cell has one read port and one write port. An array of thirty-two words of 32 cells was used (total 1024 bits). The return stack stores the return address when a call instruction is executed, this fact permits a very fast sub-routine return mechanism.

In the return stack's design, a basic six transistor static CMOS memory cell was used [5]. In this case the PMOS-load transistors are small, since they act only as a latch, and must not drive the bit line. Due to the fact that the bit line are precharged to one, the NMOS pull-down transistor was designed properly in order to drive fast enough these line.

The address and control blocks of the return stack consist of an adder/sub, a 5 to 32 bits decoder and a read/write logic. Flags from the add/sub block trigger the appropriate trap instruction in the control unit. See figure 2(a).

B. Operand Stack

The operand stack is used to store operands, to call arguments or to hold some values, it is interconnected by three buses to other elements of the operative part, for this reason the instruction set is based on a triadic model.

From the control section of the operand stack, two big blocks can be observed, first one allows partial random access to the stack, so, like as [6], it is possible simultaneously to access the top position and one of the three adjacent locations. The stack pointer contains the address of the most recent stored item, and three pointers more indicate the second (S_{n-1}), third (S_{n-2}) and fourth (S_{n-3}) locations down, related to the stack pointer.

The possibility to access the top four positions, leads to a single-cycle implementation for some of the Forth stack instructions. So, this means that one primitive of the language can frequently be implemented with one instruction. So, if a instruction will be executed, the first data may be at the top stack location, the second will either one, or two, or three locations down, and the result data is then stored in one of these positions.

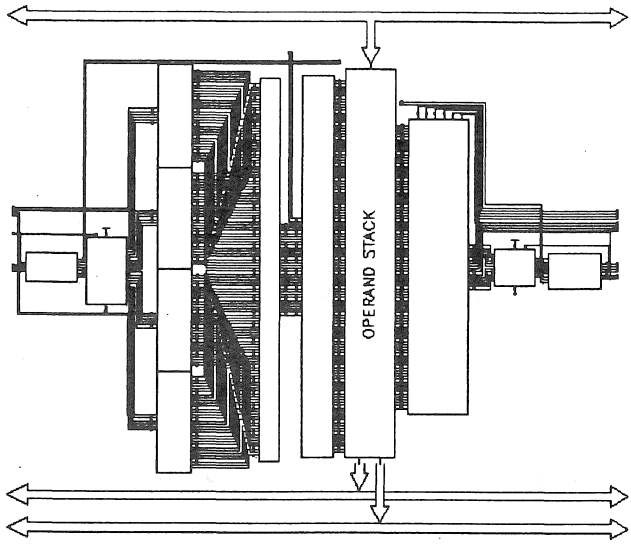


Figure 2. Operand stack block diagram.

Two read commands permit simultaneously transfer two operands from the stack through two buses to the operator unit. In a stack oriented architectures the result of an operation is transferred from the operator through another bus to the stack. By the way, all instructions use as data two of the last four operands stored into the stack.

Furthermore, the possibility to shift or rotate one position, the data in the four top locations in just one clock cycle, represents the main feature of this implementation. As a matter of fact, this structure allows to execute FORTH composite instructions in one instruction cycle.

The shift/rotate operations have a variable depth and can operate on two, three or four items. The control logic has to active only those pointers which define the appropriate locations. The second block of the control logic select the appropriate locations to execute the variable-depth rotate operation.. The stack was implemented as a variable-four-position shift register function.

The push operation causes the stack pointer to increment and a pop causes the pointer to decrement. The stack pointer is incremented/decremented through an adder/sub block, starting from the S_1 first position. This block generates the overflow and the underflow flags in the S_n and S_1 positions, respectively.

The size of the operand stack has been studied through benchmark programs. From this analysis, a stack size of 32 words would reduce the stack overflows to less than 1% [7]. Each of the registers used in the data stack requires a cell with one write port and two read ports connected to three buses. In the next item, the basic cell used is described.

IV. The operand stack customized Cell

The cell design was based on the traditional cross-coupled inverter with pass transistors inserted between them. Therefore, it consists of two separate left and right shift registers, each one using recirculating-type cells. A bidirectional shift register is very simple to implement in a MOS technology thanks to the possibility of using pass transistor [8]. The pass transistors allow to shift up or shift down the stored data into the stack. Furthermore, the cell has three ports with access to three buses architecture [9]. So, two-read ports and one-write port were considered. See figure 3(a) for more details.

The operand stack employs a three port CMOS cell using a weak inverter and two read bit lines. A good feature of this cell is that simultaneous read operations on the stack can be performed. Each port of the cell has an independent word select, global control, data and bit lines. See figure 3(b).

Due to the regular structure of the array, a well defined layout strategy of the global control, power and ground lines for the stack system can be achieved. All the word lines are running perpendicular to the data lines as is shown in figure 4(b).

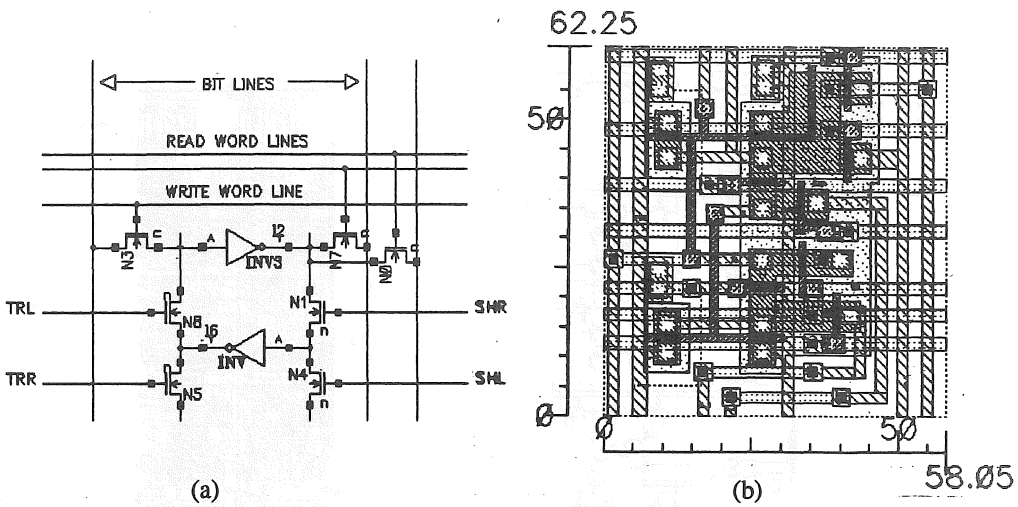


Figure 3. Operand stack basic cell. (a) Electrical diagram. (b) Layout

A. Read-Write Operations

The microprocessor global control is clocked by four non overlapping phases, the cell allows to execute the read-write operations in two consecutive phases of the cycle instruction. Due to the fact that write operation is executed by writing data stored in the bit line, only one N-ytpe pass transistor is involved in the operation, so, to transfer the data into the recirculating-type cell, the write and appropriate row select signals must be activated.

When a read operation is performed, an independient precharge pulse for each bit line is generated. By precharging the bit lines to one-level, only one NMOS transistor to access the cell is required, and only the N-type of the inverter has to be large enough to drive the long bit lines to zero. The PMOS load of the inverters can be kept small.

In the read-write operations those pass transistors which interconnect the memory cells remain inactive and the stack operates as a conventional RAM structure.

B. Shift Operations

In this structure, four additional control signals for each bit cell are required for executing the shift operations. The SHR(Shift Right), SHL(Shift Left), TRR(Transfer Right) and TRL(Transfer Left) signals are controlled to perform the rotate operations. When a shift down operation is being executed, the TRL, SHR and SHL signals are turned off and TRR is turned on. In the next cycle, TRL and SHR are turned on, TRR and SHL are inactived. The shift-rotate and read operations could not be excuted simultaneously, for this reason, three phases of clock are required to execute some instructions .

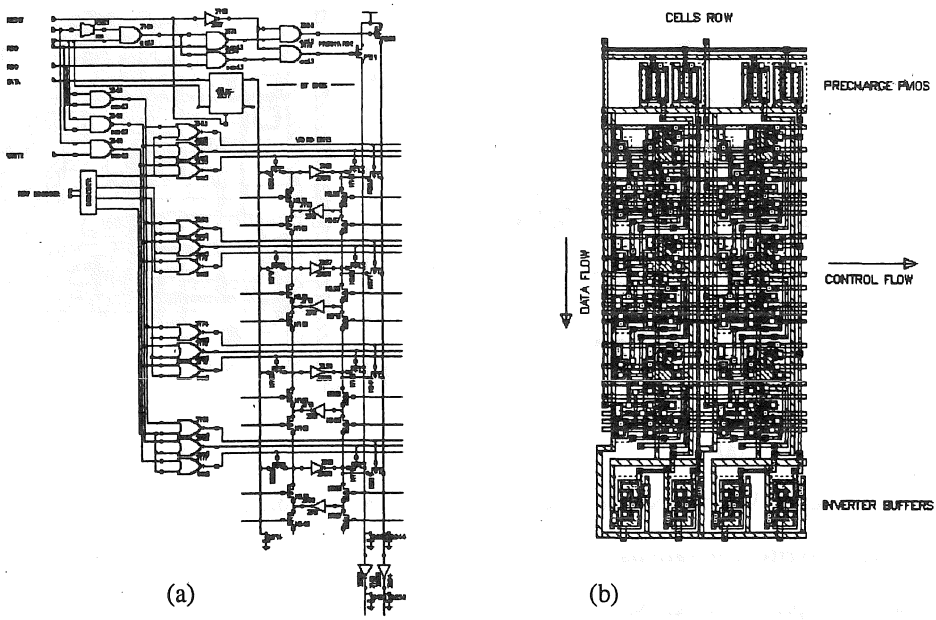
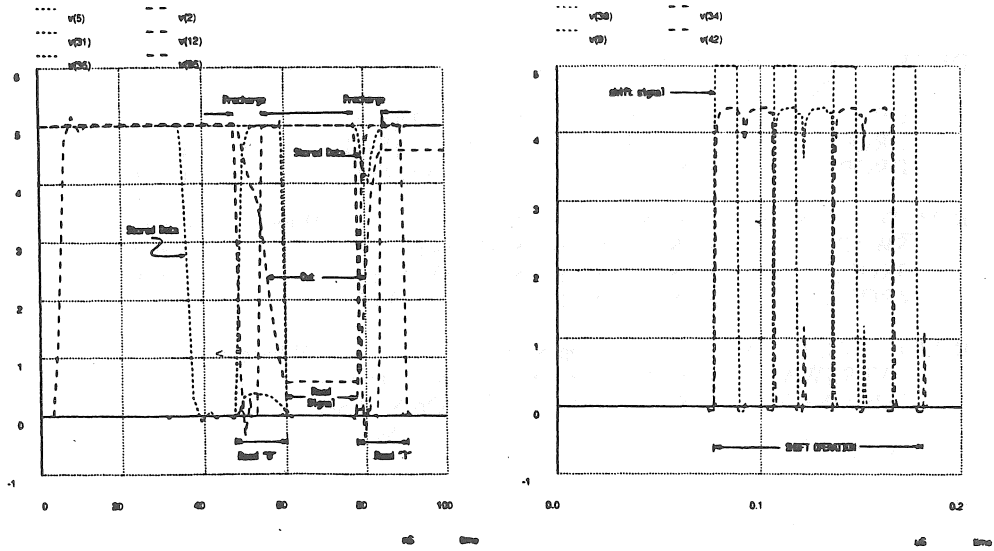


Figure 4. (a) Simulation circuit. (b) Layout strategy

Simulations were done using the SPICE 3D2 electrical simulator considering a bit line capacitances of 1.4P, the circuit used to simulate the cell is shown in figure 4(a). In figure 5(a) the results of the electrical simulation for write, precharge and read operations are shown. Figure 5(b), depicts the simulation results for shift operations.



(a) (b)

Figure 5. Simulation results. (a) Read-write operation. (b) Shift operation

V. The Instruction Set

In this section, instruction set to be executed by the stack system is described. In this visualization method we consider that the item on the rigth denote the top of the stack (S_n), S_{n-1} represents the location directly below it and so on. An single manipulation on the operation stack can simplify a program and decrease his execution time. The instruction set will be executed by controlling the stack pointer and the shift control signals. The stack system can execute both unary and binary instructions.

Unary Instructions

DUP(i) ($S_1,S_2.....S_n$ ---> $S_1,S_2.....S_n,Op[S_i]$).		Where $n-3 \leq i \leq n$
DUP	($S_1,S_2....S_n$ --- $S_1,S_2,...S_n,Op[S_n]$)	$i = n$
OVER1	($S_1,S_2....S_n$ --- $S_1,S_2,...S_n,Op[S_{n-1}]$)	$i = n-1$
OVER2	($S_1,S_2....S_n$ --- $S_1,S_2,...S_n,Op[S_{n-2}]$)	$i = n-2$
OVER3	($S_1,S_2....S_n$ --- $S_1,S_2,...S_n,Op[S_{n-3}]$)	$i = n-3$

PICK(i) ($S_1,S_2.....S_n$ ---> $S_1,S_2.....S_{n-1},Op[S_i]$)		Where $n-3 \leq i \leq n$
OVERSWAP	($S_1,S_2....S_n$ ---> $S_1,S_2....S_{n-1},Op[S_n]$)	$i = n$
DROPDUP	($S_1,S_2....S_n$ ---> $S_1,S_2....S_{n-1},Op[S_{n-1}]$)	$i = n-1$
PICK2	($S_1,S_2....S_n$ ---> $S_1,S_2....S_{n-1},Op[S_{n-2}]$)	$i = n-2$
PICK3	($S_1,S_2....S_n$ ---> $S_1,S_2....S_{n-1},Op[S_{n-3}]$)	$i = n-3$

DROP(i) ($S_1,S_2.....S_n$ ---> $S_1,S_2....S_{n-2},Op[S_i]$). Where $n-3 \leq i \leq n$

SWAPDUP	($S_1,S_2....S_n$ ---> $S_1,S_2....S_{n-2},Op[S_n]$)	$i = n$
DROP	($S_1,S_2....S_n$ ---> $S_1,S_2....S_{n-2},Op[S_{n-1}]$)	$i = n-1$
DROP2	($S_1,S_2....S_n$ ---> $S_1,S_2....S_{n-2},Op[S_{n-2}]$)	$i = n-2$
DROP3	($S_1,S_2....S_n$ ---> $S_1,S_2....S_{n-2},Op[S_{n-3}]$)	$i = n-3$

ROT_D(i) ($S_1,S_2..S_n$ ---> $S_1..S_{i-1},S_{i+1}..S_n,Op[S_i]$) Where $n-3 \leq i \leq n-1$

SWAP	($S_1..S_n$ --> $S_1..S_{n-2},S_n,Op[S_{n-1}]$).	$i = n-1$
3ROT_D	($S_1..S_n$ --> $S_1..S_{n-3},S_{n-1},S_n,Op[S_{n-2}]$).	$i = n-2$
4ROT_D	($S_1..Sn$ --> $S_1..S_{n-4},S_{n-2},S_{n-1},S_n,Op[S_{n-3}]$)	$i = n-3$

ROT(i) ($S_1,S_2...S_n$ ---> $S_1..S_{i-1},Op[S_i],S_{i+1}..$) Where $n-3 \leq i \leq n$

OVERSWAP	($S_1,S_2..S_n$ --> $S_1..S_{n-3},S_{n-2},S_{n-1},Op[S_n]$)	$i = n$
2SWAPOVER	($S_1,S_2..S_n$ --> $S_1..S_{n-3},S_{n-2},Op[S_{n-1}],S_n$)	$i = n-1$
3ROT	($S_1,S_2..S_n$ --> $S_1..S_{n-3},Op[S_{n-2}],S_{n-1},S_n$)	$i = n-2$
4ROT	($S_1,S_2..S_n$ --> $S_1..Op[S_{n-3}],S_{n-2},S_{n-1},S_n$)	$i = n-3$

ROT_U(i) ($S_1...S_n$ ---> $S_1..S_{i-1},Op[S_n],S_i..S_{n-1}$) Where $n-3 \leq i \leq n-1$

SWAPR	($S_1,S_2..S_n$ --> $S_1..S_{n-3},S_{n-2},Op[S_n],S_{n-1}$)	$i = n-1$
3ROT_U	($S_1,S_2..S_n$ --> $S_1..S_{n-3},Op[S_n],S_{n-2},S_{n-1}$)	$i = n-2$
4ROT_U	($S_1,S_2..S_n$ --> $S_1..Op[S_n],S_{n-3},S_{n-2},S_{n-1}$)	$i = n-3$

SHF(i) ($S_1...S_n$ ---> $S_1...Op[S_i],S_{n-3},S_{i\pm1},S_n$) Where $n-2 \leq i \leq n-1$

SHF2	($S_1,S_2...S_n$ ---> $S_1..Op[S_{n-1}],S_{n-3},S_{n-2},S_n$)	$i = n-1$
SHF3	($S_1,S_2...S_n$ ---> $S_1..Op[S_{n-2}],S_{n-3},S_{n-1},S_n$)	$i = n-2$

The **binary instructions** only add one second operand to the operation. The instruction cycle is not increased.

VI. Conclusions

The implementation of a stack cache for an application-specific in the FORTH directed microprocessor with stack-based architecture, has been presented. The operand stack uses a cell with three port access which allows shift up or down the data stored in the stack.

Data are transferred directly into the stack operand through one write port. The two read ports allow simultaneously get two operands and carry-out to the microprocessor operational units.

Furthermore, the possibility to shift the data one position up or down, is a main feature of this implementation. The presented solution proved to be very efficient to perform composite FORTH instructions in one instruction cycle. The stack system presented was designed in 1.2 µm CMOS technology, the address and control blocks were designed using the standard cell library from ES2 silicon foundry. The size of the cell was 59 µm x 63 µm and it provides a highly regular structure leading to a well defined layout strategy.

VIII. References

[1] P.J. Koopman, Jr, Stack Computers-The new wave, Ellis Horwood Limited, John Wiley & sons, New York.

[2] W. P. Salman, O Tisserand, B. Toulout, "FORTH", Macmillan Computer Science Series, Editions EYROLLES, París, 1983.

[3] J.S. James, What is FORTH ? A tutorial Introduction, Byte Publications, August, 1980.

[4] P.E. Danielsson, A Variable-Length Shift-Register, Correspondence, IEEE Transactions on Computers, Vol c-32, No 11, November, 1983.

[5] K. O'Connor, The Twin-Port Memory Cell, IEEE Journal of Solid States Circuits, Vol sc-22, No 5, October, 1987.

[6] S.C. Lee, J.R. Hayes, Development of a FORTH Language Directed Processor using Very Large Scale Integrated Circuitry, John Hopkins APL Technical Digest, Vol 10,Number 3, 1989.

[7] D. L. Miller, Stack Machines and Compiler Design, Byte Publications, April, 1987.

[8] A. Mukherjee, "Introduction to NMOS and CMOS VLSI System Design", Prentice Hall, 1986.

[9] M. L. Anido, D. J. Allerton, E. J. Zaluska, A three-Port/Three-access Register File for Concurrent Processing and I/O Communication in a RISC-LIKE Graphics Engine, ACM, 1989.

Extensión del Modelo de Sistemas Expertos usando Aprendizaje Automático

Sacha A. Bocic David A. Fuller

Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Casilla 306
Santiago 22
Chile

Resumen

En este artículo se presenta la incorporación de técnicas de aprendizaje automático al área de sistemas expertos. Para ello, se propone una extensión al modelo tradicional de sistemas expertos que consiste en incorporar una interfaz inteligente. Dicha interfaz se basa en técnicas de aprendizaje automático, con la capacidad de adquirir conocimiento durante la interacción con el usuario. Se presentan dos interfaces, para sistemas expertos con razonamiento con encadenamiento hacia adelante y hacia atrás. Para ambas interfaces se muestran ejemplos e implementaciones en PROLOG.

Palabras Clave: Aprendizaje Automático, Sistemas Expertos, EBG, Interfaz Inteligente.

1. Introducción

Se ha demostrado que los sistemas expertos son herramientas muy útiles para resolver problemas en muchos campos, tales como en planificación y en medicina. Sin embargo, esta tecnología presenta inconvenientes al momento de ser aplicada para resolver problemas de la vida real. Estos inconvenientes los encontramos en las diferentes etapas del ciclo de vida de un sistema experto. Dentro de los problemas más significativos, figuran aquellos relacionados con la etapa de desarrollo y aquellos relacionados con la etapa de utilización. En la etapa de desarrollo, tenemos la dificultad de extraer el conocimiento del experto como una fuerte limitante [7,13,15]. Por otro lado, en la etapa de utilización de un sistema experto, la velocidad de respuesta constituye un serio problema en algunos casos.

Un sistema basado en conocimientos puede efectuar un proceso aprendizaje de dos tipos o niveles [11]. El aprendizaje a nivel del conocimiento consiste en incorporar al sistema, conocimiento externo que no es cubierto por la clausura del conocimiento[1]. El segundo tipo de aprendizaje corresponde a la definición de aprendizaje en [14], es decir, a cambios en el sistemas que son adaptivos en el sentido que permiten al sistema realizar exactamente la misma tarea, sacada de la misma colección de preguntas, en forma más eficiente la próxima vez. En [8] y [12] se presentan trabajos relacionados con la adquisición automática de conocimiento, es decir, con el primer tipo de aprendizaje. En este artículo se presentan técnicas de aprendizaje del segundo tipo, aplicadas al área de sistemas expertos.

Usualmente, en los sistemas expertos encontramos mecanismos de razonamiento que son una mezcla de razonamiento con encadenamiento hacia adelante y hacia atrás[2]. Un ejemplo típico de este tipo de sistema son los de diagnóstico. En efecto, si nos trasladamos al terreno de la medicina, es común que el médico se encuentre con un conjunto de síntomas (antecedentes) a partir de los cuales intenta probar una hipótesis (para lo cual tiene que recurrir a razonamiento con encadenamiento hacia atrás), y así prueba un diagnóstico (conclusión).

Por ejemplo, si se tiene el conjunto de reglas de la figura 1 y se observan los síntomas `pulso_debil` y `taquicardia`, se intenta probar `presion_baja` para concluir `shock`. Así, `presion_baja` será probado con razonamiento con encadenamiento hacia atrás, para luego concluir, usando razonamiento con encadenamiento hacia adelante la condición de `shock`.

1.- La "clausura de conocimientos" corresponde al conocimiento que el sistema puede deducir a partir del conocimiento que se representa en forma explícita en la base de conocimientos.

2.- En inglés, "forward" y "backward chaining", respectivamente.

```
if (pulso_debil and taquicardia and presion_baja) then shock
if (presion_sistolica < 90) then presion_baja
```

Figura 1. Ejemplo de reglas de un sistema de diagnóstico.

Para los efectos de este artículo, hemos decidido tratar los dos tipos de razonamiento por separado para facilitar la comprensión de los distintos aspectos involucrados en cada uno de ellos. Por este motivo, se presentan dos interfaces inteligentes para sistemas expertos, cuyas principales características son la de contar con la capacidad de analizar una consulta, extraer información y almacenarla para su posterior uso. Se asume que las interfaces forman parte de un sistema experto implementado en PROLOG, por lo tanto los ejemplos se presentan en dicho lenguaje. Si el lector no está familiarizado con técnicas de programación y meta-programación en PROLOG, puede consultar [17,1].

2. Modelo Clásico de Sistema Experto

Un modelo sencillo de un sistemas experto consta de cuatro componentes principales: la base de conocimiento, la máquina de inferencia, el generador de explicaciones y la interfaz con el usuario (figura 2).

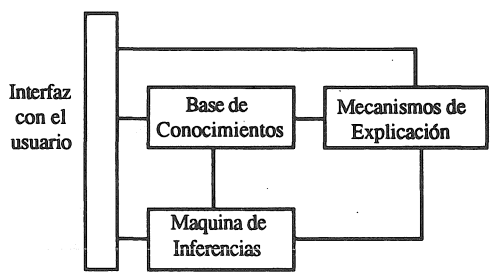


Figura 2. Modelo simple de un Sistema Experto.

La base de conocimientos es el conjunto de información acerca de los objetos y de las relaciones entre estos objetos dentro de un dominio específico y acotado. Existen muchas formas de representar este conocimiento, siendo las más comunes las reglas, los marcos[3] y las redes semánticas. En nuestro caso, lo haremos mediante reglas. La función de la máquina de inferencia es obtener conclusiones a partir de la base de conocimiento. Existen varias estrategias para realizar lo anterior y a modo de ejemplo, podemos mencionar el mecanismo de resolución de PROLOG.

3.- Más conocidos por la palabra en inglés "frames".

Todo sistema experto debe ser capaz de explicar como llega a una conclusión y porqué hace una pregunta al usuario, entregándo información adicional al usuario para que éste entienda lo que se le está preguntando para poder contestar correctamente. Estas tareas las cumple el módulo de mecanismos de explicación. Por último, la interfaz permite comunicar al usuario con la máquina de inferencia y con la base de conocimientos.

3. Extensión al Modelo de Sistemas Expertos

Una de las principales falencias de los sistemas expertos convencionales radica en que éstos no se benefician de la experiencia pasada [15]. Por ejemplo, consideremos un sistemas experto de diagnóstico que frente a un paciente, su historial clínico y síntomas, determina que el paciente sufre de una enfermedad X. Si a este mismo sistema experto se le presenta otro paciente con una historia clínica y síntomas similares, también diagnosticará la enfermedad X porque habrá *repetido* todo el proceso.

Por otro lado, es bastante acéptada la idea que un experto humano es básicamente una persona con mucha experiencia en un área determinada. Por lo tanto, el no considerar este factor en un sistema experto no parece ser razonable. Esto nos hace pensar en la posibilidad de que el sistema se beneficie de la experiencia anterior de tal forma de no tener que repetir todo el proceso de inferencia, logrando un mayor grado de eficiencia. Esto se traduce, en último término, en una mayor velocidad de respuesta del sistema.

Un sistema experto que se beneficia de la experiencia debe ser capaz de analizar casos y extraer información útil para el futuro. Así como también, debe reconocer casos similares a los analizados en el pasado y resolverlos utilizando la información almacenada para tal efecto. En este artículo se propone incorporar al modelo clásico de sistema experto, una interfaz inteligente, cuya función es la descrita. El modelo propuesto se presenta en la figura 3.

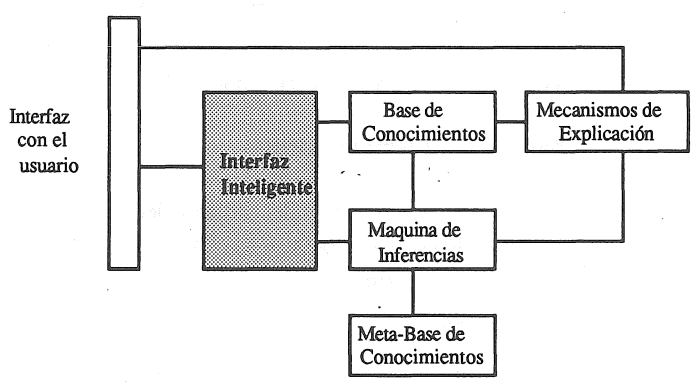


Figura 3. Modelo propuesto de Sistema Experto.

Está claro que la interfaz debe contar con algún tipo de mecanismo de aprendizaje (para extraer y almacenar información) y de razonamiento por analogía (para reconocer casos similares). Existe un gran número de técnicas de aprendizaje automático que pueden ser aplicadas en el área de sistemas expertos [13], sin embargo, al seleccionar el mecanismo de aprendizaje, se debe tener en cuenta el tipo de razonamiento que hace el sistema (con encadenamiento hacia adelante, hacia atrás o combinaciones de ellos) y la forma en que se representa el conocimiento.

Considerando que el conocimiento lo representamos por medio de reglas, es posible recurrir a una técnica de aprendizaje deductivo conocida como Generalización Basada en Ejemplos o más conocida por su sigla en inglés EBG [10]. Esta técnica es muy apropiada para aplicarla en razonamiento con encadenamiento hacia atrás. Para razonamiento con encadenamiento hacia adelante se propone una técnica basada en la anterior. El mecanismo de razonamiento por analogía es bastante primitivo y consiste en considerar similar dos consultas si una de ellas es subsumible a otra. En las siguientes secciones se presentan implementaciones y ejemplos de dos interfaces inteligentes.

4. Interfaz Inteligente para Razonamiento con Encadenamiento hacia Atrás

El razonamiento con encadenamiento hacia atrás parte desde una hipótesis que queremos demostrar y llega a las condiciones necesarias para que ésta sea verdadera, que pueden ser hechos de la base de conocimientos o reglas cuya validez debe ser demostrada. En términos de un árbol de prueba, esto lo podemos describir como la construcción del árbol desde la raíz hacia las hojas.

Ejemplos típicos de sistemas expertos con este tipo de razonamiento son los de planificación. En este tipo de sistema experto, un problema se modela con un estado inicial, un estado final y un conjunto de acciones u operadores. Cada operador o acción se define en términos de sus precondiciones y efectos. Un plan es una secuencia de operadores, que al ser aplicados en orden, permiten alcanzar el estado final a partir del estado inicial [5].

4.1. Generalización Basada en Ejemplos (EBG)

EBG es definido en [10] como "un método independiente del dominio para guiar la generalización usando conocimiento dependiente del dominio". Para los efectos de este artículo, se presenta una descripción de un sistema EBG como parte de un sistema experto con razonamiento con encadenamiento hacia atrás.

Bajo esta perspectiva, podemos decir que los cuatro parámetros de entrada que recibe todo sistema EBG los "toma" del sistema experto. El primer parámetro, el **concepto ob-**

jetivo o lo que se desea aprender, es una hipótesis que el sistema experto recibe para intentar demostrar, pero usando variables libres. El segundo parámetro es el llamado **ejemplo de entrenamiento**. En términos de sistemas expertos, corresponde a toda la información necesaria para que el sistema experto pueda demostrar una instancia del concepto objetivo, es decir, pueda demostrar que la hipótesis que recibe el sistema experto con las variables instanciadas es verdadera.

Para probar que el ejemplo de entrenamiento es una instancia del concepto objetivo, EBG construye un árbol de prueba [9] que tiene como nodo raíz a la hipótesis instanciada. Este árbol se llama "árbol del ejemplo de entrenamiento". Paralelamente se construye el "árbol generalizado", que es un árbol de prueba que tiene como nodo raíz al concepto objetivo. Para construir ambos árboles de prueba se utiliza el conjunto de reglas y hechos que provee la base de conocimientos del sistema experto. La base de conocimientos corresponde a la **teoría de dominio**, es decir, al tercer parámetro de entrada a EBG.

Ambos árboles se construyen idénticos (para cada nodo que se expande, ambos árboles seleccionan la misma regla) hasta alcanzar los predicados "operacionales". El "**criterio operacional**" determina si un predicado es o no operacional. La definición de este criterio es dependiente del dominio, por lo tanto, depende del sistema experto. Este criterio corresponde al cuarto parámetro de entrada a EBG. Usualmente se definen como operacionales, a los predicados que representan una pregunta al usuario, los hechos (facts) y aquellos predicados fácil de calcular.

Cuando en una rama del árbol del ejemplo de entrenamiento se alcanza un predicado operacional, se suspende la expansión de la rama correspondiente al árbol generalizado y se expande completamente la que corresponde al árbol del ejemplo de entrenamiento.

Por último, el proceso de reformulación no es más que la conjunción de todos los nodos hoja del árbol generalizado, es decir, de todos los nodos operacionales del árbol generalizado. De esta forma se obtiene una nueva definición para el concepto objetivo pero que es más fácil de calcular que la original (por lo tanto más eficiente).

A modo de ejemplo, consideremos el conjunto de reglas de la figura 4. Para este caso, definimos como operacional a los predicados: `u(X)`, `t(X)` y `v(X)` y supondremos que además se tiene la suficiente información para demostrar `p(a,b)`, es decir, es posible construir un árbol de prueba completo para `p(a,b)`.

```
p(X,Y):- q(X),r(Y).          u(X):- ...
q(X):- s(X),t(X).            v(X):- ...
s(X):- u(X).                 t(a).
r(Y):- v(Y).                 t(b).
.....
```

Figura 4. Ejemplo de reglas para EBG.

En la figura 5, el árbol de la izquierda corresponde al del ejemplo de entrenamiento, es decir, `p(a,b)` y el de la derecha es el árbol generalizado para el concepto objetivo `p(X,Y)`. La reformulación que se obtiene es `p(X,Y):- u(X),t(X),v(Y)`, que corresponde a la regla aprendida por el sistema.

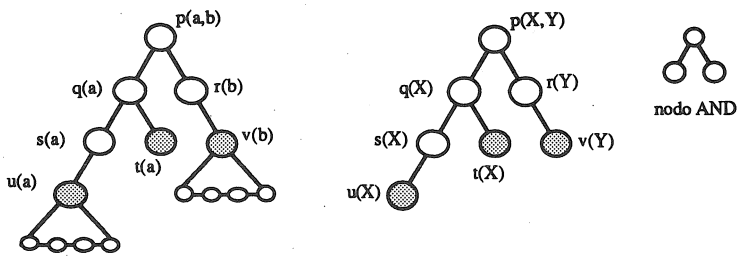


Figura 5. Arboles de prueba para las reglas de la figura 4.

4.2. Interfaz Para Razonamiento con Encadenamiento Hacia Atrás

EBG recibe como entrada un concepto objetivo, una teoría de dominio, criterio operacional y ejemplo de entrenamiento y entrega como salida una reformulación para el concepto objetivo en términos de predicados operacionales.

La idea más útil que aporta EBG es la de reformular un predicado en términos de otros más fácil de calcular. Es así como podemos pensar en una interfaz inteligente que ante cada hipótesis sugerida, chequea si ha sido procesada una hipótesis similar en el pasado, de tal forma de usar la información que se tiene almacenada para tal efecto. Si no ha sido procesada, la interfaz debe tomar la hipótesis sugerida (reemplazando las constantes por variables libres) como concepto objetivo y generar una reformulación para dicha consulta, de tal forma que en el futuro se pueda contestar la misma consulta en forma más eficiente. El ejemplo de entrenamiento lo constituye, en este caso, el conjunto de predicados operacionales instanciados que se utilizan para la demostración de la hipótesis. Se asume que el criterio operacional ha sido definido. En [2] se presenta una implementación de una interfaz inteligente con las características mencionadas pero basada en una extensión a EBG [2] que permite generar reformulaciones más eficientes.

4.2. Ejemplo

A modo de ejemplo consideremos el clásico problema de planificación del *mono y las bananas* . Este problema consiste en que se tiene un mono en una pieza y una banana colgada del techo. El mono debe alcanzar la banana, para ello cuenta con una serie de objetos (mesa, silla, garrote, etc.) con los cuales se puede ayudar. El objetivo es generar una secuencia de acciones para que el mono logre su objetivo. La solución se presenta en la figura 6 en forma de un programa en PROLOG, la que consideraremos (para los efectos de este artículo) como un sistema experto de planificación.

```
plan(A,B,Plan):- sameXY(A,B,P), sameZ(A,B,[],P,Plan).

sameXY(A,B,P):- position(A,From), position(B,To),
                P=['move ',A,'from:',From, 'to:', To].

sameZ(A,B,OldObjs,Pi,P):- height(Obj,Hp), height([B],Hb), Hp<Hb,
                          new(OldObjs,NewObjs,Obj),NewP=['Get:',Obj],
                          append(Pi,NewP,Po),sameZ(A,B,NewObjs,Po,P).

sameZ(_,B,OldObjs,P,P):- height(OldObjs,Hp), height(B,Hb), Hp>=Hb.

height([],0).

height(Target,H):- target(Target,H).

height([X|Xs],N):- h(X,N0), height(Xs,N1), N is N1 + N0.

new(Olds,News,Obj):- object(Obj,_),\+ member(Obj,Olds)[4],
                     append([Obj],Olds,News)[5].
```

Figura 6. Ejemplo del mono y las bananas.

En esta solución, el objeto que se instancia en el primer argumento logra alcanzar al objeto instanciado en el segundo, si ejecuta la secuencia de planes que se instancia en el tercer argumento. Luego, si consideremos el conjunto de hechos de la figura 7 como un ejemplo de entrenamiento y como concepto objetivo la consulta `plan(A,B,Plan)`, para utilizar la interfaz inteligente sólo falta definir el criterio operacional.

```
object(mono,5).      position(mono,(0,0)).
target(banana,16).   position(banana,(12,21)).
object(caja,8).      object(silla,6).
```

Figura 7. Ejemplo de entrenamiento.

Antes de definir el criterio operacional, es necesario aclarar que para un mismo sistema es posible definir más de un criterio operacional. También es importante notar que la definición

---

4.- El predicado `member/2` es verdadero si el término que se instancia en el priemer parámetro pertenecea a la lista que se instancia en el segundo parámetro.

5.- El predicado `append/3` es verdadero si el tercer argumento es sintácticamente igual a la concatenación de los dos primeros.

del criterio es una decisión de diseño que tiene directa repercusión sobre la calidad de la reformulación y a la sobrecarga asociada a la interfaz inteligente. Una vez aclarado lo anterior, definimos como operacional para la teoría de dominio de la figura 6 a los predicados: `pos/2`, `height/2`, `new/3` y los operadores aritméticos '<' y '>='.

Luego podemos ocupar la interfaz inteligente en el ambiente PROLOG y hacer la consulta `interfaz(plan(mono,banana,Plan))`, (el predicado `interfaz/1` define la interfaz) con lo que se obtiene como resultado la instanciación de la variable `Plan` (en el plan requerido) y como efecto lateral se le agrega a la base de conocimientos la regla de la figura 8. Además se agrega a la meta-base de conocimientos información que indica que existe información para la consulta recién procesada.

```
plan(A,B,['move:',A,' from:',P1,' to:',P2,' get:',Obj1,' get:',Obj2]):-
        pos(A,P1), pos(B,P2), height([A],H1), height(B,H2), H1 < H2,
        new([A],Objs1,Obj1), height(Objs1,H3), height(B,H4), H3 < H4,
        new(Objs1,Objs2,Obj2), height(Objs2,H5), height(B,H6),H5 >= H6.
```

Figura 8. Reformulación de una regla utilizando la Interfaz Inteligente.

De esta forma, la siguiente vez que se quiera obtener un plan para un objeto "A", primero se intentará utilizar la nueva reformulación, si esta falla, se efectuará un proceso de aprendizaje basado en la nueva consulta.

5. Interfaz Inteligente para Razonamiento con Encadenamiento hacia Adelante

El razonamiento con encadenamiento hacia adelante consiste en derivar una conclusión a partir de los antecedentes. En términos de árboles de prueba, esto consiste en partir la construcción desde las hojas (antecedentes) para llegar a la raíz (conclusión). El mecanismo de aprendizaje que se propone para esta interfaz está basado en EBG, pero con la diferencia que construiremos el árbol de prueba desde las hojas hacia la raíz. Este mecanismo, que llamamos EBG_BU (por EBG Bottom Up), recibe como parámetros de entrada una **teoría de dominio** y un **ejemplo de entrenamiento** en forma de un conjunto de antecedentes (que tienen las variables instanciadas). Estos dos parámetros son comparables a la teoría de dominio y ejemplo de entrenamiento de EBG. El otro parámetro que recibe EBG_BU le llamamos **antecedentes generalizados**, que corresponde al mismo conjunto de literales del ejemplo de entrenamiento, pero con la sustitución de las variables instanciadas por variables libres. Este parámetro es análogo al concepto objetivo de EBG.

El mecanismo comienza seleccionando reglas en cuyos cuerpos aparecen los literales del ejemplo de entrenamiento. Cada vez que encuentra una regla en cuyo cuerpo aparecen solamente literales del ejemplo de entrenamiento, se efectúa un proceso de reducción, que consiste en eliminar del ejemplo de entrenamiento los literales que aparecen en el cuerpo de la regla y

agregar a dicho conjunto la cabeza de la regla. Es así como se genera el árbol de prueba en forma ascendente. Paralelamente, se mantiene una copia del árbol que se genera, en el cual las variables que aparecen instanciadas en el ejemplo de entrenamiento se reemplazan por variables libres (árbol generalizado).

El último parámetro que recibe es el **criterio de detención**, que es análogo al criterio operacional de EBG, es decir, determina hasta que punto se construye el árbol generalizado. Para los efecto de este artículo, se considera que el árbol de prueba se construye hasta la raíz, es decir, completo. Una vez que se termina la construcción del árbol, se genera la reformulación que no es más que una regla que tiene el nodo raíz como consecuente y los nodos hoja como antecedente.

A modo de ejemplo, consideremos el conjunto de reglas de la figura 4 y supongamos que el criterio de detención nos dice que la construcción se detiene al llegar al nodo raíz y que tenemos como punto de partida `u(a)`, `t(a)` y `v(b)`. La reformulación es: `p(X,Y):- u(X),t(X),v(Y)`. Esto se muestra en la figura 9.

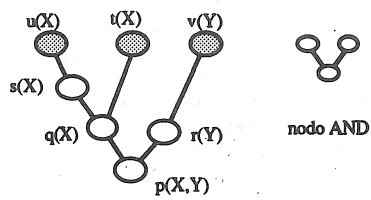


Figura 9. Ejemplo de reformulación bottom-up.

5.1. Ejemplo

A modo de ejemplo, consideremos el conjunto de reglas de la figura 10. Estas reglas representan una parte de un sistema experto para el diagnóstico de la metrorragia (sangramiento de los genitales) durante la segunda mitad del embarazo. Una implementación de una interfaz inteligente para razonamiento con encadenamiento hacia adelante se presenta en [2].

```
diagnostico(Paciente,ru):- sintomas(Paciente,ru),
                           historiaClinica(Paciente,ru),
                           examenMedico(Paciente,ru).

sintomas(Paciente,ru):- she[6](Paciente,positivo),
                        inicioSangramiento(Paciente,brusco).

historiaClinica(Paciente,ru):- cicatriz(Paciente,presente).
```

6.- "she" es una abreviación de Síndrome Hipertenso del Embarazo.

```
examenMedico(Paciente,ru):- hemorragia(Paciente,mixta),
                            shock(Paciente,positivo),
                            dolor(Paciente,presente),
                            feto(Paciente,Palpable).

hemorragia(Paciente,mixta):- hemorragia(Paciente,externa),
                             hemorragia(Paciente,interna).

shock(Paciente,positivo):- presion(Paciente,baja),
                           taquicardia(Paciente,presente),
                           colorPiel(Paciente,palido),
                           pulso(Paciente,debil).

presion(Paciente,baja):- presionSistolica(menorQue90).

taquicardia(Paciente,presente):- frecuenciaPulso(mayorQue100).
```

Figura 10. Reglas de Sistema Experto de Diagnóstico de la Metrorragia de la Segunda Mitad del Embarazo.

En la figura, sólo es posible derivar un diagnóstico (rompimiento uterino o "ru"), el cual depende de los síntomas e historia clínica de la paciente, además del examen médico que se le efectúe. Ahora, podemos utilizar la interfaz inteligente (definida por el predicado `interfaz_bu`) de la siguiente forma:

```
?- interfaz_bu((colorPiel(ana,palido), pulso(ana,debil), she(ana,positivo),
                hemorragia(ana,externa),feto(ana,palpable),
                hemorragia(ana,interna), frecuenciaPulso(ana,mayorQue100),
                inicioSangramiento(ana,brusco),
                cicatriz(ana,presente),dolor(ana,presente),
                presionSistolica(ana,menorQue90)),Head).
```

con lo que se obtiene como solución la instanciación del segundo argumento de `interfaz_bu/2` en `diagnostico(ana,ru)`. Como efecto lateral se agrega a la meta-base de conocimientos información que indica que existe información para la consulta recién procesada y la regla recientemente aprendida (que se muestra en la figura 11) se agrega a la base de conocimientos.

```
diagnostico(P,ru):- colorPiel(P,palido), pulso(P,debil), she(P,positivo),
                    hemorragia(P,externa), feto(P,palpable),
                    hemorragia(P,interna), frecuenciaPulso(P,mayorQue100),
                    inicioSangramiento(P,brusco), cicatriz(P,presente),
                    presionSistolica(P,menorQue90), dolor(P,presente).
```

Figura 11. Resultado de la reformulación usando EBG_BU.

6. Conclusiones

En este artículo hemos presentado dos interfaces inteligentes, una para razonamiento con encadenamiento hacia adelante y otra para razonamiento con encadenamiento hacia atrás. La utilidad y funcionalidad de ambas interfaces se mostró mediante ejemplos simplificados de sistemas expertos de las áreas de diagnóstico y planificación.

La ventaja de la utilización de estas interfaces radica en el incremento en la velocidad de respuesta que obtiene el sistema. Este incremento se produce como consecuencia de efectuar un proceso de aprendizaje (según la definición de [14]), al no tener que construir todo un árbol de prueba (cuando se tiene información sobre la consulta), ya que se utiliza una regla que es equivalente, pero más fácil de calcular. Además, se evita intentar probar ramas fallidas en el proceso de construcción del árbol de prueba La magnitud de este beneficio depende de la forma del espacio de búsqueda que define el sistema experto. Así por ejemplo, en un espacio profundo y angosto, el beneficio es más significativo que en uno ancho y bajo, como se aprecia en la figura 12.

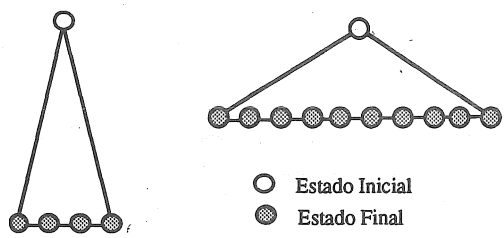


Figura 12. Tipos de espacios de búsqueda.

Un caso que no se ha tratado es el de las múltiples reformulaciones. En efecto, para una consulta es posible obtener más de una reformulación, lo que supone generar más de una regla. En [6] se muestra como EBG puede llevarnos a un caso de "explosión disjuntiva", en el cual la eficiencia del sistema se degrada al agregar nuevas reglas. Distintas formas de atacar este problema se presentan en [2].

Un punto que hay que considerar es que la incorporación de una interfaz inteligente implica una sobrecarga al sistema. En efecto, en términos de meta-programación, se habla de agregar una capa más al sistema y como se sabe, cada capa degrada el rendimiento en un orden de magnitud [17]. En las dos interfaces mostradas, la sobrecarga afecta al sistema durante el proceso de aprendizaje, es decir, cuando no hay información acerca de la consulta. Cuando se tiene información, la consulta es procesada por el intérprete de PROLOG, por lo tanto, la sobrecarga es eliminada. En consecuencia, la utilización de interfaces inteligentes es adecuada para sistemas en los cuales el número de procesos de aprendizaje es mucho menor que el de procesamiento de consultas sobre las cuales se tiene información.

Por otro lado, vemos que el proceso más importante que realizan ambas interfaces consiste en la construcción de un árbol de prueba. En la medida que este proceso sea eficiente, la interfaz inteligente significará menos sobrecarga al sistema total. En [2,3,4] se presentan extensiones a los algoritmos clásicos de EBG y evaluación parcial, cuya incorporación puede mejorar considerablemente la eficiencia de las interfaces.

Dado que es posible definir los criterios operacional, de reflectabilidad y de detención de muchas formas para un mismo sistema experto y considerando que estas definiciones repercuten en la calidad de las reformulaciones y en la sobrecarga del sistema, se propone como trabajo futuro el estudio de algún mecanismo que permita generar dichas definiciones en forma automática.

También se propone como trabajo futuro el diseño e implementación de una interfaz inteligente para razonamiento mixto (hacia adelante y atrás) ya que en la práctica, casos en los cuales sólo hay un tipo de razonamiento es difícil de encontrar. Por último, se propone como trabajo futuro la incorporación de aprendizaje inductivo a interfaces inteligentes de sistemas expertos.

Agradecimientos

Deseamos manifestar nuestro agradecimiento al Dr. Andrés Kychentall por su valiosa colaboración en el sistema experto de diagnóstico de la metrorragia de la segunda mitad del embarazo.

Este trabajo ha sido financiado gracias al apoyo del Fondo Nacional de Ciencia y Tecnología, FONDECYT, proyectos 90-0688 y 89-0591 y a los Proyectos de Fomento a la Investigación 91/21 y 90/8 de la Dirección de Investigación de la Pontificia Universidad Católica de Chile.

Referencias

[1] I. Bratko, Prolog Programming for Artificial Intelligence, Wokingham et al.: Addison Wesley (2nd Edition) (1990).

[2] S. Bocic, Manipulación Simbólica en Sistemas Basados en Conocimiento, Tesis de Magister, P. Universidad Católica de Chile, Santiago, (1992).

[3] D. Fuller, S. Bocic, "Un Experimento en Aprendizaje Automático y Evaluación Parcial". En actas de la XVII Centro Latinoamericano de Estudios en Informática (CLEI), Caracas, Venezuela, (1991).

[4] D. Fuller, S. Bocic, "Extending Partial Evaluation in Logic Programming", COMPUTER SCIENCE: Research and Applications, pp. 95-107, Plenum Press, N.Y., (1992).

[5] H. Hendler, A. Tate, M. Drummond, "AI Planning: Systems and Techniques". AI Magazine, pp. 61-77, Verano (1990).

[6] H. Hoppe, "An Analysis of EBG and its Relation to Partial Evaluation: Lessons Learned", Reporte técnico No. 572, GMD - IPSI, Alemania, Sept. (1991).

[7] P. Jackson, Introduction to Expert Systems, Addison Wesley, (1986).

[8] J. Kornell, "Automated Knowledge Acquisition and Representation", Reporte técnico, General Research Corporation, Santa Barbara, CA, Estados Unidos, (1984).

[9] J. Lloyd, Foundations of Logic Programming, Springer-Verlag, (1984).

[10] T.M. Mitchell, R.M. Keller, S.T. Kedar-Cabelli, "Explanation-based Generalization: A unifying view", Machine Learning Vol I, (1986).

[11] A. Newell, "The Knowledge Level", Artificial Intelligence, No. 18, pp. 87-127, (1982) .

[12] M.A. Newstead, R. Pettipher, "Knowledge Acquisition for Expert Systems", Electrical Communication, Vol. 60, No 2, pp. 115-121, (1986) .

[13] R. Reich, S. Fenves, "The Potential of Machine Learning Techniques for Expert Systems". AI EDAM, pp. 175-193, (1989).

[14]. H.A. Simon, Why Should Machines Learn?, en Machine Learning: An Artificial Intelligence Approach, R.S. Michalski, J.G. Carbonell y T.M. Mitchell (Eds), (1983).

[15] S. Slade, "Case-Based Reasoning: A Research Paradigm", AI Magazine, pp. 42-55, Primavera (1991).

[16] L. Sterling, R.D. Beer, "Incremental Flavor-Mixing of Meta-Interpreters for Expert Systems Construction", en Proc. 3rd Symposium on Logic Programming, pp.20-27, Salt Lake City, Estados Unidos, (1986).

[17] L. Sterling, E. Shapiro, The Art of Prolog. The MIT Press, Cambridge, Massachusetts, Londres, Inglaterra, (1986).

Prototyping Action Semantics Using Functional Languages

Paulo Borba* Silvio Meira† André Santos‡

Departamento de Informática
Universidade Federal de Pernambuco

Abstract

Action Semantics is a development of Denotational Semantics used to write formal descriptions of programming language semantics.

In this work we describe an extension to a lazy functional language (Lazy ML) to support Action Semantics notation, so that one can (almost) directly translate Action Semantics specifications of programming languages into executable prototypes of their interpreters.

Lazy ML's action notation extension proved to be a very useful tool in the process of validation of specifications, as it makes possible to rapidly obtain a prototype interpreter of a language directly from its specification. We describe also how the system was used to produce a prototype interpreter for a functional language.

Keywords

Action Semantics, Semantic Prototyping, Semantics of Programming. Languages

1 Introduction

The formal description of a programming language is an essential document for its development, given its many important roles: it is a guide to the implementation, documents design decisions, establishes standards to be followed by implementations, provides the basis to prove program correctness in what is concerned to its specification, may be used as a reference manual for the language, etc. Informal descriptions are not as useful, for they are usually incomplete, inconsistent and ambiguous.

The syntax of programming languages is traditionally described in a formal way using context free grammars. For the definition (specification) of their semantics there are many formalisms: Operational [5], Axiomatic [2] and Denotational [6]. These formalisms,

*This author's current address:Department of Computing Science, Oxford University, e-mail: Paulo.Borba@prg.oxford.ac.uk

†e-mail: silvio@di.ufpe.br. Work of this author partly funded by CNPq grant 300133/85-CC, CNPq, Brazil. The other two authors are supported by CNPq research scholarships.

‡This author's current address:Department of Computing Science, Glasgow University, e-mail: andre@dcs.glasgow.ac.uk

initially used in theoretical studies, were not very successful when used to describe "real" programming languages like Pascal or ADA. In these cases the resulting descriptions were incomplete or extremely long and very hard to understand.

Action Semantics [4] was developed to make possible the formal specification of "real" programming languages, taking care of many of the problems in the formalisms described above. For that end, pragmatic aspects were considered: readability, modularity and reusability of the specification. These aspects are easily seen in the example we will show, specially if compared to other formalisms like denotational semantics.

Another very important point when developing a programming language is that of obtaining a prototype compiler or interpreter for the language as soon as possible. This is important for validating the language definition and for finding errors in the specification. Although Action Semantics descriptions are very operational and intuitive, there is no formal and direct way to transform a specification into a running prototype. This can be done in an entirely ad hoc way, where one cannot assure the correctness of the transformation process, and possibly loosing the level of abstraction one would like to have while specifying a language. In this case the effort to get the prototype will change the major issue of the work from the language itself to its implementation.

If it were possible to obtain a running interpreter directly from the Action Semantics specification of the language, it would be much easier and faster to try new characteristics of the programming language, and possibly improve them, what would be harder with a prototype developed in a different way. This way the specification would be in direct correspondence with the implementation and its changes could be immediately reflected in a new version of the interpreter. The cost of the continuous development of a prototype would be much reduced.

In this work we describe the implementation and use of an Action Semantics interpreter. It consists of a series of modules written in Lazy ML [1] implementing the Action Semantics notation. Other lazy functional languages could be used as well, as we use only purely functional constructs, that are present in most functional languages. This way, one can rapidly and with minimal effort write a specification and have it compiled and running. It has proven to be very useful as a tool for verifying the syntactic and type correctness of Action Semantics descriptions (there are no tools available for as yet), as well as a rapid way of getting an interpreter for a language from its formal definition, and to validate the specification.

A similar approach to the same problem is shown in [7] describing how to implement a subset of an action notation similar to that described in [4]. The main difference from our approach is that there it is not described how handle sorts neither support for specific semantic entities. In [8] a prototyping method is presented to solve these problems, and differs from our work basically by having a more complete sort handling mechanism. In both works the prototyping system is written in Standard ML, and uses specific characteristics of the language not available in other functional languages, like exception handling. Our approach was developed independently of both works and uses purely functional constructs.

We initially introduce Action Semantics, then describe part of its notation and show how the library of action notation functions was built, concluding by detailing a method for translating Action Semantics descriptions into executable prototypes. Then we describe how we translated the original action semantics description of a functional language to obtain directly from its specification the prototype of a running interpreter written in the action notation extension of Lazy ML.

2 Action Semantics

Action Semantics is a development of Denotational Semantics used to write formal descriptions of programming language semantics. This goal is achieved through modularity and readability of the formal descriptions.

Action Semantics is denotational (or "compositional") in that the semantics of each phrase of the language (expressions, declarations, etc.) is expressed in terms of the semantics of its subphrases.

The main difference between Action and Denotational Semantics is in the entities used to give meaning to programs. Denotational Semantics uses high order functions, expressed in lambda notation and Action Semantics uses special entities called *actions*, which have a (reasonably) simple operational interpretation and quite nice algebraic properties.

The Action Notation is used to express the basic actions to compute new values, bind tokens to values, etc., and action combinators that are used to build complex actions and denote sequential execution, action selection, etc. The generality of these combinators, which follow the suitable algebraic laws, provides the inherent modularity of the semantic descriptions. Thanks to the use of commonly used words with their intuitive meanings, actions expressed in the notation are intentionally verbose and suggestive, making it possible to gain a (broad) impression of a language's semantics from a superficial reading of its Action Semantics description.

Actions can be performed, and they process information gradually, in a step-by-step way. Therefore, they are inherently more operational than functions. When performed, an action (which may be part of an enclosing action) can:

- *complete*, corresponding to normal termination (the performance of the enclosing action proceeds normally) or
- *escape*, corresponding to exceptional termination (the enclosed action is skipped until the escape is trapped) or
- *fail*, corresponding to abandoning the performance of an action (the enclosing action performs an alternative action, if there is one, otherwise it fails too) or
- *diverge*, corresponding to non termination (the enclosing action also diverges).

Considering the performance of an action as corresponding to the behaviour of a program, actions can be used to represent the semantics of programs. Moreover, actions can be used to represent the contribution that parts of the program give to the semantics of the whole program. This is expressed through the mapping of programming language phrases to the actions they denote.

The information processed by actions are elements (and possibly sorts) of data. Abstractions are used to encapsulate actions and are classified as data as well, although actions themselves are not.

The various kinds of information give rise to different "facets" of the actions, according to the kind of information they process at a time:

- *control* facet, which ignores information processing.
  Ex: $\boxed{A_1 \text{ and then } A_2}$ is the action that executes A_1, and if A_1 finishes A_2 is executed. A_1 and A_2 are actions.

- *functional* facet, processing transient information (actions are given and give data). Transient information consists of simple data, possibly tagged, corresponding to intermediate results.
  Ex: $\boxed{A_1 \text{ then } A_2}$ is the action that executes A_1, and then executes A_2; A_1 receives the data the compound action receives, A_2 receives the data generated by A_1, and the data generated by the compound action are those generated by A_2.

- *declarative* facet, processing scoped information (actions receive and produce bindings). Scoped information consists of bindings of tokens to data, corresponding to symbol tables.
  Ex: $\boxed{A_1 \text{ before } A_2}$ is the action that executes A_1, and then executes A_2; A_1 receives the bindings received by the compound action. A_2 also receives the bindings produced by A_1, A_1 having precedence. The bindings produced by the compound action are the ones produced by A_1 and A_2, A_2 having the precedence.

- *imperative* facet, processing stable information (actions reserve and unreserve cells, and change the data stored in the cells). Stable information consists of data stored in cells, corresponding to values assigned to variables.
  Ex: $\boxed{\text{store } D_1 \text{ in } D_2}$ is the action that stores the datum D_1 in the cell D_2.

For space reasons, it is not possible to give a detailed account of Action Semantics here.

The semantic description style we use follows that used in [4], where the abstract syntax of the language is defined first, then the semantic entities that will express the semantic values of programs, and finally the semantic functions, expressed by actions that map the phrases of the abstract syntax to semantic values.

The notation used to specify the abstract syntax is a form of context-free grammar, similar to BNF.

The Semantic Entities module describes the semantic domains that are used, to whom the programs will be mapped, and operations over these domains (*sorts*).

In Semantic Functions we describe the meaning of a program through the mapping of its abstract syntax to semantic entities.

3 Action Semantics Notation in Lazy ML

The action notation in Lazy ML is structured as a library of predefined functions and data types defined as Lazy ML modules.

Basic data (sets, maps, etc.) are defined as abstract data types, and their respective operations (union, dom, etc.) defined as functions over them.

The basic, functional, declarative and imperative actions (combinators) are defined as high order functions that receive other actions (functions) and information about the state of transient, scoped and stable information (the state before the performance of the action) as input and give behaviours (the state after the action performance plus its status, e.g. complete, fail or escape) as result. Another possible implementation for actions could be to implement them as algebraic data type with its combinators and primitive actions as constructors. In this approach, instead of directly compiling the specification we would define a function to interpret the combinators during execution.

The state before and after action performance consists of a 4-tuple containing the information about the state of the evaluation. The 4-tuple represents the transient, scoped, stable and control information.

The transient information component can contain the three types of information that can be given to (or given by) an action from the functional facet point of view:

- Non-tagged information, meaning that the action that receives this information is given data with no tags, like in **give** D, where D is a datum.
- A set of tagged information, meaning that the action that receives this information is given data with tags, like **give** D **tag**[n]. The set is represented as a mapping from tags (n, represented as integers) to data (D).
- No transient information, meaning that the action receives no information.

In the scoped information field we have a mapping from tokens to data, representing the current binding information.

The stable information field contains the status of the storage (memory cells). It contains a mapping from cells (addresses) to data, or an empty mapping meaning that no cells were reserved, in case there is no storage information.

Finally the control information contains auxiliary information that is used internally in the implementation, having no equivalent in the action semantics definition.

An action behaviour contains the same kind of information as the state plus information whether it has completed, escaped or failed.

The basic semantic entities (basic data) of action notation were implemented as constructors of an algebraic data type named `datum`, mimicking the **datum** semantic entity of action notation. In reality, any new semantic entities that are included in a specification must be defined adding a new constructor to the ones already defined.

abstraction is the sort of data that encapsulates actions. The encapsulated action is performed when the abstraction is enacted. In the implementation it is represented by the `Abstraction` constructor, that has an `Act` (that is, a function with type `(Info -> Behaviour)`) as its component.

As an example of how Action Semantics combinators (actions) are implemented receiving the state information and giving an action behaviour, let us see how the **give** functional action is expressed in Lazy ML. **give** D is an indivisible action that completes, giving the datum D, failing when D yields '**nothing**'. This action is implemented as follows:

```
give dat (info as (t,sc,st,c)) =
  let d = eval dat info
  in if d = Nothing then Fail (NoTransInfo,NoScInfo,st,NoConInfo)
                    else Complete (NonTagged d,NoScInfo,st,NoConInfo)
```

where `datum` is a dependent datum that is evaluated with the current information (`info`) yielding `d`. When `d` is `Nothing` (**nothing**) the action fails with the received storage information and no other information. Otherwise, the action completes with the received storage information and the transient information `NonTagged d`.

All operators are defined similarly, that is, as receiving the same parameters as in action notation plus the state information. Then, if it receives a datum, the datum is evaluated with the current information and the behaviour of the action is expressed as a `Behaviour` with the respective changes in the state information. If the action receives other actions (subactions) as parameters its evaluation depends on the evaluation of its subactions, so they are evaluated with information corresponding to the behaviour of the enclosing action. The implementation of the **then** functional action is an example:

```
(a1 then' a2) (i as (t,sc,st,c)) =
    case (a1 i) in
      Complete (t1,sc1,st1,c1) :
        case (a2 (t1,sc,st1,c)) in
```

```
        Complete (t2,sc2,st2,c2) :
          Complete (t2,MScInfo sc sc1 sc2,st2,MConInfo c c1 c2)
      || Escape (t2,sc2,st2,c2) : Escape (t2,sc2,st2,MConInfo c c1 c2)
      || Fail (t2,sc2,st2,c2) : Fail (t2,sc2,st2,MConInfo c c1 c2)
      end
|| any : any
end
```

It is defined as a Lazy ML function that receives two actions `a1` and `a2` (that is two functions, possibly the `then'` itself with two other actions as arguments), and the state information `i` with its four components. The action `a1` is evaluated with the state information received by the `then'` combinator. If `a1` fails or escapes, the whole action behaves the same way, and `a2` is not even evaluated (the same happens if `a1` diverges). If the evaluation (its behaviour) completes, `a2` is then evaluated with the state information `(t1,sc,st1,c)`, that is, with the transient and storage information generated by `a1` and the scoped and control information given to the `then'` combinator. If the evaluation of `a2` completes, the behaviour of the action `a1 then' a2` is also `Complete` with state information composed by the transient and storage information generated by `a2` (`t2` and `st2`), and scoped and control information that is a combination of those received by the `then'` combinator and those generated by `a1` and `a2` (`MScInfo sc sc1 sc2` and `MConInfo c c1 c2`). If `a2` escapes or fails `a1 then' a2` also escapes or fails with transient, scoped and storage information generated by `a2`, and a combination of control information generated by `a1` and `a2` and received by the `then'` combinator. If `a2` diverges the whole action also diverges. The other combinators are similarly implemented.

4 From Action Semantics to an Interpreter

The method for transforming an Action Semantics description in a working interpreter prototype in Lazy ML consists, in most cases, of a simple syntactic translation. Some minor changes are needed to handle new semantic entities the description might contain in a proper way. The modules of the action semantics notation can be implemented as Lazy ML modules, to keep the implementation as modular as the specification.

Syntax. The abstract syntax is translated into a series of algebraic data types that will function as the language syntax thereafter. It is quite straightforward to derive an algebraic data type from the syntax.

Semantic Entities. The semantic entities defined as basic data in action notation are already implemented as constructors of the `datum` algebraic data type. New semantic entities that are specific to the language being defined must be included as new constructors for `datum`, if needed. If new constructors are introduced in `datum`, some modifications must also be made in the definition of functions which compare semantic entities, e.g. the equality operator must be modified to be able to compare the new entities.

Semantic Functions. The semantic functions are implemented as functions from the abstract syntax (defined as an algebraic data type) to an action. That is achieved by defining functions by pattern matching on the data types used as abstract syntax. The function body is identical to the original semantic function's body with minor syntactic changes. Let us take as an example a semantic function from the action semantics description of a subset of a lazy functional language under development in our department. We can, by comparing the definition of the semantic function **evaluate** $[\![E_1\ E_2]\!]$ in action notation and in Lazy ML's action notation. **evaluate** $[\![E_1\ E_2]\!]$ checks whether E_1 is an abstraction (in

this case the denotation of a function) and gives the result of its application to the argument E_2, or a constructor, giving the corresponding construction. It is defined in action notation as follows:

```
evaluate ⟦E1 E2⟧ =
    | evaluate ⟦E1⟧
    then before
    | | | give closure (abstraction(evaluate ⟦E2⟧)) tag [2]
    | | and then
    | | | give the resulting value tag [1]
    | then
    | | | check the resulting value[1] is an abstraction
    | | and then
    | | | enact the abstraction[1] with the argument value[2]
    | | or
    | | | check the resulting value[1] is a Value-Constructor
    | | and then
    | | | give construction(the Value-Constructor[1], the argument value[2]) ;
```

In the Lazy ML version of action notation this definition is transformed into

```
evaluate (ES e1 e2) =
    evaluate e1
  then
      ((givetag (closure (Abstraction (evaluate e2))) 2
      andthen
        givetag (the (resulting value)) 1)
    then'
        ((check ((thetagged (resulting value) 1) is' (an abstraction))
        andthen
          enact ((thetagged abstraction 1) with
                 (thetagged (argument value) 2)))
      or
         (check ((thetagged (resulting value) 1) is' (a ValueCon))
        andthen
          give (construction (thetagged ValueCon 1,
                              thetagged (argument value) 2)))))
```

Most of the semantic functions change as little as this one when redefined in Lazy ML.

5 Conclusions

The Action Semantics description of a language is an important document for its development and use, and can be very helpful in verifying correctness of implementations. It is much more readable than descriptions given in other formal methods such as denotational semantics and the specifications are inherently modular. This makes them highly reusable, because many language constructs have identical meaning in many programming languages.

The Action Semantics description of a programming language, together with a working prototype of an interpreter for the language that is obtained directly from its semantic

description, can be used to try new ideas in the design of the language and find errors much earlier in the process of its development.

Many simple (syntax and typing) errors were found in the original specification of the example language of this project when we first tried to compile it, demonstrating the usefulness of the system. They were easily found when using the system, and would be very difficult to find without it. These problems could be easily solved if there were syntax and type checkers for Action Semantics specifications.

Although the process of transforming the original Action Semantics description in one proper for compilation is not formal, the proximity of the two notations (where most definitions need minor syntactic changes) makes the process of transformation very natural and intuitive.

It should be possible to formalise the transformation step and build a syntax directed translator that will make most of the transformation, translating the original specification syntactically to the compiler notation, preserving its semantics.

Another important feature will be to support the Action Semantics communicative facet. That was not possible due to difficulties in handling processes in functional languages like Lazy ML. An implementation of the same modules in a functional language supporting processes (e.g. the functional language A), seems to be possible [3].

Acknowledgements

Many thanks are due to Professor Peter Mosses, who most kindly sent us his lecture notes on-the-fly, for a course in Recife in the fall term 1989.

Hermano Moura suggested modifications and made very interesting comments on an early draft of this paper.

References

[1] L. Augustsson and T. Johnsson. *Lazy ML User's Manual.* Department of Computer Science, Chalmers University of Technology, Göteborg, draft edition, 1988.

[2] J. A. Goguen. Parameterized programming. *IEEE Trans. on Software Eng.*, pages 528–543, September 1984.

[3] S. R. L. Meira. Processes and functions. In *TAPSOFT'89*, number 351 in LNCS, Barcelona, 1989. Springer-Verlag.

[4] P. D. Mosses and D. A. Watt. The use of Action Semantics. In M. Wirsing, editor, *Conference on Formal Description of Programming Concepts III.* IFIP TC2, North-Holland, Amsterdan, 1987.

[5] G. D. Plotkin. A structural approach to operational semantics. Technical report, Aarhus University, September 1981.

[6] D. A. Schmidt. *Denotational Semantics.* Allyn and Bacon, 1986.

[7] D. A. Watt. Executable semantic descriptions. *Software—Practice and Experience*, 16(1), January 1986.

[8] D. A. Watt. *Programming Language Syntax and Semantics.* Prentice-Hall International, 1991.

SUPORTE DE BANCOS DE DADOS PARA SISTEMAS DE HIPERTEXTOS

Marcos Roberto S. Borges

Núcleo de Computação Eletrônica
e
Instituto de Matemática

Universidade Federal do Rio de Janeiro
Caixa Postal 2324
20001 - Rio de Janeiro
e-mail: borges@ufrj.bitnet

RESUMO

Sistemas de hipertextos e de bancos de dados guardam entre si uma relação muito estreita no que diz respeito ao armazenamento e tratamento de grandes volumes de informação. Apesar disto, muito poucas das soluções adotadas para bancos de dados tem sido adotadas em sistemas de hipertextos, o que parece um contracenso, já que um hipertexto é um banco de dados. Este artigo estuda esta questão e analisa como as soluções já consolidadas na área de bancos de dados podem ser incorporadas a uma arquitetura de sistemas de hipertextos.

ABSTRACT

Hipertext and database systems have much in common in respect to storage and management of large volumes of data. In spite of that, few are the solutions already adopted in databases that are being applied to hypertext systems. This article addresses this issue and describes how these solutions can be applied to hypertext systems. It also analyses the adoption of an architecture for hypertext systems similar to that proposed by ANSI/SPARC to database management systems.

Palavras-chave: Hipertextos, Hiperbase, Bancos de Dados

I - INTRODUÇÃO

Hipertextos/Hipermidia são termos que hoje já se consolidaram como denominação de uma área própria da ciência da computação. A ideia é considerada antiga [5] mas só recentemente tem recebido maior atenção por parte de pesquisadores. Vários sistemas de diferentes portes e características foram lançados no mercado e muitos mais protótipos experimentais estão em desenvolvimento principalmente em universidades e centros de pesquisa [6, 11, 12, 13].

Esta perspectiva de disseminação generalizada do uso começou a levantar questões sobre os diversos aspectos da arquitetura e implementação de tais sistemas, em especial no suporte ao tratamento de grandes volumes de dados e na interface homem máquina, fatores considerados cruciais no desenvolvimento de sistemas de hipertextos/hipermídia [8, 9, 10, 16].

Há obviamente uma grande diferença entre a implementação de sistemas de uso pessoal, quase experimental, em micro-computadores e aqueles destinados a um uso mais real, ou seja multi-usuário e/ou para máquinas de maior porte como estações de trabalho e até mainframes, em especial no aspecto do tratamento de grandes volumes de informação. Assim, é de se esperar uma grande correlação entre as áreas de hipertextos/hipermídia e bancos de dados [18].

Em princípio, a correspondência das duas áreas pode ser vista como total. Sendo a área de bancos de dados mais consolidada, era de se esperar que muitas das soluções propostas e adotadas para bancos de dados fossem aproveitadas em sistemas de hipertexto [17]. Isto entretanto não acontece. Há alguns sistemas inclusive que abandonaram o uso de um SGBD para suporte ao sistema de hipertexto preterindo-os por sistemas convencionais de arquivos.

Este artigo pretende examinar uma questão que parece ser das mais importantes quando se analisa a arquitetura de sistemas de hipertexto: A correspondência dos problemas de armazenamento e tratamento da informação com sistemas de gerenciamento de bancos de dados. Aspectos mais consolidados como as soluções adotadas para os controles operacionais - concorrência, restrições de integridade, recuperação de falhas e segurança - são avaliados.

A motivação principal deste artigo é exatamente a definição de uma arquitetura para um sistema de hipertexto de suporte ao Projeto HIPERBASE [4] dentro do ambiente PRADES [1, 2, 3] em desenvolvimento na UFRJ.

O restante do artigo se divide em 3 outras partes. Na primeira examinamos brevemente alguns aspectos de hipertextos. Na segunda discutimos a adaptação das soluções consolidadas em bancos de dados para a sistemas de hipertexto e algumas das alternativas em estudo no Projeto HIPERBASE. Na última parte apresentamos as conclusões do trabalho.

II - CARACTERÍSTICAS DE SISTEMAS DE HIPERTEXTOS/HIPERMÍDIA

Hipertextos tem se desenvolvido como uma área separada da informática, embora esteja claro que ela empresta conceitos de diversas outras áreas já consolidadas. Uma boa definição de hipertexto foi dada por Norman Meyrowitz na palestra de abertura da conferência Hypertext '89 [15]: "*Hipertexto é uma encruzilhada onde vários interesses se encontram*".

Sistemas de hipertextos podem ser "vistos" como um grande repositório de informações de diversos tipos: texto, código executável, gráficos, imagens estáticas ou animadas e som; fragmentadas segundo algum critério. Os repositórios podem ser chamados de

hiperdocumentos e o particionamento de nós. Uma outra parte é uma estrutura ou um conjunto de estruturas de ligação entre estes nós, denominadas de ligações. Como as ligações entre os nós são do tipo M x N torna-se necessário definir mais um elemento básico: os botões que representam funções de associação entre trechos de um nó a outro nó. A Figura 1 ilustra um exemplo de um hiperdocumento.

Dentro de um processo de criação de um hiperdocumento há a etapa que é denominada de autoria que consiste das operações: definição e geração do conteúdo dos nós; marcação dos botões de ligação em cada nó; definição dos nós de destino de cada ligação.

Há ainda dois outros conceitos que fazem parte do processo de autoria: a definição de versões e a definição de visões ou contextos [14]. A primeira permite que se possua num mesmo hiperdocumento várias versões de um mesmo nó. A segunda permite diferentes visões do hiperdocumento como um todo, possibilitando, por exemplo, que somente parte dos nós, botões e ligações possam ser percorridos.

A visão ou contexto de um hiperdocumento tem por objetivo apresentar os nós (e suas ligações) que correspondem a um determinado contexto do hiperdocumento. Ao contrário de bancos de dados entretanto, as visões em hipertextos não permitem operar sobre resultados de combinações ou restrições.

Nós e ligações podem possuir ainda um conjunto de descritores que armazenam dados sobre o tipo de nó, o autor, a data de criação, o número da versão, as visões dos quais participam e outros mais.

Um outro modo de operação de um sistema de hipertexto é aquele que permite ao usuário somente navegar pela estrutura sem poderes para altera-la. Normalmente, quando um usuário entra no ambiente é lhe oferecido um primeiro nó (nó de abertura). A partir daí, a navegação se dá por ativação de um dos botões acionados, por exemplo, por um "click" no "mouse". É comum também oferecer alternativas de desvio comuns a todos os nós, através de um menu de botões que desviam para nós de índice ou mapas.

Podemos verificar nesta fase do artigo que é essencial para um sistema de hipertexto um poderoso suporte de armazenamento de dados, seja isto oferecido por um sistema próprio de arquivos ou um sistema gerenciador de bancos de dados. A correspondência entre os problemas e suas possíveis soluções são os temas abordados na próxima seção.

III - SOLUÇÕES DE BANCOS DE DADOS PARA SISTEMAS DE HIPERTEXTO

Apesar de muitos dos problemas e das soluções adotadas para sistemas de hipertexto serem genuínas, há muita interseção com a área de banco de dados, pois ambas se destinam ao tratamento das informações. Nesta seção procuraremos analisar os aspectos do projeto de sistemas de hipertexto que possuem uma solução correspondente em sistemas de bancos de dados e verificar como esta solução se adaptaria a sistemas de hipertexto.

Quando se projeta um sistema cogitando-se apoiar o armazenamento de dados do sistema de hipertexto em um SGBD, é necessário uma análise mais aprofundada das facilidades oferecidas do SGBD, normalmente projetado para sistemas convencionais, onde a informação é naturalmente estruturada. A flexibilidade do enfoque de hipertextos contrasta com esta rigidez e faz com que muitos projetos abandonem já de início esta possibilidade.

A análise de sistemas de hipertextos aqui realizada se concentra em 4 aspectos principais a saber: a arquitetura do sistema, a modelagem do esquema conceitual do hiperdocumento, os mecanismos de busca e a análise dos controles operacionais.

III.1 - Arquitetura do Sistema

Desde 1978 quando foi inicialmente apresentada a proposta da ANSI/SPARC para uma arquitetura de 3 níveis para sistemas de gerenciamento de banco de dados, passou-se a considerar esta arquitetura como ideal porque proporcionava um isolamento de cada uma das 3 camadas: externa, conceitual e interna, das demais. Assim, no nível conceitual, voltado para o administrador do banco de dados, os objetos e seus relacionamentos são tratados isoladamente dos aspectos de máquina e de cada usuário; no nível interno são definidas as estruturas de armazenamento visando a eficiência da recuperação e atualização dos dados; e finalmente no nível externo são definidas as visões externas de cada grupo de usuários.

Tal como estão definidos a maioria dos sistemas de hipertexto, só há um nível de interação, e em alguns casos com dois enfoques: o da autoria e o de leitura/navegação. Os aspectos conceituais e de armazenamento ou não são visíveis ou são tratados conjuntamente com a definição de nós e ligações.

Propõe-se portanto, uma arquitetura de três níveis, semelhante à adotada em sistemas de bancos de dados a fim de isolar os aspectos conceituais, do usuário específico da máquina, alcançando a chamada independência dos dados. Como em sistemas de hipertexto é usual a separação dos leitores dos autores, sugere-se então a definição dessas duas classes no nível externo. Um esquema mostrando a arquitetura proposta é apresentado na Figura 2.

A partir dessa arquitetura, o ponto de partida para a definição de um hiperdocumento passa a ser a definição conceitual da estrutura do mesmo, na qual a abstração dos tipos de nós e ligações é feita. Para isto é necessário um modelo de dados, assunto discutido a seguir.

III.2 - Modelagem de Dados

Conforme mencionado anteriormente, os elementos básicos para estruturação da informação em um sistema de hipertexto são os nós e as ligações. O resultado desta simplicidade é uma grande liberdade por parte do autor na fragmentação da informação, sem qualquer preocupação com um modelo estrutural pré-definido. Por outro lado, embora possamos identificar facilmente a existência de elementos estruturais como nós e ligações, torna-se difícil capturar a semântica destas mesmas ligações ou do conteúdo dos nós. A estrutura conceitual do enfoque de hipertexto é portanto implícita e não há separação do que é estrutura e da instância dessa estrutura.

Em contraste, sistemas de bancos de dados obrigam a estruturação completa da informação forçando que a limitação de dados à base se dê sob regras rígidas, definidas mo processo de modelagem de dados. Essa estruturação proporciona uma plataforma para descrever as instâncias na forma de entidades e relacionamentos genéricos e também para a formulação de consultas.

O que propomos é a adoção do mesmo enfoque para sistemas de hipertexto, mas possibilitando e não obrigando a descrição dos tipos de nós e ligações baseada no significado do seu conteúdo.

Algumas propostas nessa direção já foram apresentadas, embora a maioria delas visando uma aplicação específica, e portanto não possibilitando uma estruturação genérica. Um exemplo é o modelo adotado no projeto gIBIS [7]. No hipertexto gIBIS há 3 tipos de nós e 9 tipos de ligações. Esta estrutura facilita o trabalho de autoria e induz a uma melhor visualização do hiperdocumento. A organização estrutural do gIBIS é mostrada na Figura 3.

O projeto HIPERBASE [4] propõe a generalização da ideia de estruturação através da definição conceitual do hipertexto de maneira similar a de um modelo conceitual de banco de dados. Assim a tarefa de autoria é antecedida pela definição da sua estrutura, quando os tipos de nós, ligações e restrições de integridade aplicáveis são definidas. Um exemplo desta definição para um sistema de controle de cartas e propostas é mostrado na Figura 4.

A adoção deste enfoque significa que os conceitos já consolidados na área de modelagem de dados, tais como generalização, agregação e associação, podem ser utilizados em hipertextos.

III.3 - Mecanismos de Busca

Em sistemas de hipertexto, o mecanismo de busca adotado é a navegação. Como o hipertexto não possui uma estrutura conceitual, a navegação é realizada pelas instâncias dos nós, o que significa que o interface de busca em sistemas de hipertexto é a própria informação e não uma linguagem de consulta como estamos acostumados em sistemas convencionais.

Em bancos de dados convencionais, a busca da informação se dá através de uma consulta, que utiliza o esquema como uma estrutura de suporte para formulação das mesmas. Se por um lado o usuário necessita conhecer a estrutura do banco de dados, por outro, o mecanismo de consultas oferece resultados precisos, sem ambigüidade, e com expressividade semântica obtida pelo esquema conceitual do banco de dados.

Conforme sugerido por Halasz [9], o mecanismo de consulta é uma necessidade para grandes hiperdocumentos, a fim de oferecer um meio mais rápido e preciso na localização dos nós de interesse do usuário. Um sistema de hipertexto que utilize um modelo de dados, conforme sugerido na seção anterior, e adote um SGBD para o tratamento de informações pode ter a disposição todo o mecanismo de consulta do sistema, normalmente em uma linguagem padrão SQL, integrando dessa maneira as duas formas de busca: navegação e consulta. Uma extensão interessante do processo de consulta é a de integrar na linguagem meios de tratar as ligações do hiperdocumento, gerando assim uma interface adicional para lidar com as ligações [4].

III.4 - Controles Operacionais

Um outro aspecto refere-se aos controles operacionais necessários. Controle de concorrência, versões, definições e materialização de visões são apenas alguns exemplos das rotinas que devem ser implantadas a partir do zero quando não se usa um SGBD. Este não é absolutamente um trabalho trivial. Os vários anos de pesquisa destas questões na área de bancos de dados possibilitaram o desenvolvimento de sistemas razoavelmente eficientes. O abandono destas soluções pura e simplesmente não é uma solução inteligente.

Por outro lado reconhece-se a rigidez com que estes mecanismos são implantados em sistemas de bancos de dados atualmente em uso. Quase nenhuma flexibilidade (se alguma) é oferecida a quem define o banco de dados, no trato destas questões. As soluções apresentadas por sistemas existentes está muito voltada para o tratamento de dados convencionais o que torna difícil a tarefa de adapta-los à problemática de sistemas de hipertexto.

O aspecto da definição dos contextos oferecidos aos usuários na navegação se assemelha muito à definição de visões em bancos de dados. A diferença está na maneira em que uma visão é definida. Idealmente, a definição de contextos em hipertextos deveria poder contemplar também o modo em que a definição é feita em bancos de dados. Para isso entretanto, é necessário que os nós possuam tipos e descritores associados, como as entidades são definidas em bancos de dados. No projeto Hiperbase contextos são definidas da mesma forma que visões em um SGBD, isto é, através de cortes horizontais e verticais e associações de objetos. O atributo que é tratado diferentemente é o conteúdo do nó que deixa de apontar (e mostrar os botões) para outros nós que não façam parte da visão [4].

O suporte ao trabalho cooperativo é sem dúvida uma das grandes vantagens dos sistemas de hipertextos. Entretanto, para possibilitar esta condição, mecanismos de controle de concorrência se fazem necessários. Neste caso, todavia, há de se buscar soluções adotadas para bancos de dados não convencionais como CAD, que possuem transações longas e portanto necessitam de mecanismos mais flexíveis.

O aspecto de segurança é uma consequência direta do tipo de uso que se fará do hiperdocumento gerado. Até agora os tipos de aplicações propostas induz muito mais ao uso generalizado da informação do que a um acesso restritivo. Entretanto, espera-se que mecanismos de segurança, desde o nível mais trivial baseado em senhas, até aquele auxiliado pela definição de visões ou contextos restritivos, sejam necessários. Também nesta área há vários avanços alcançados na área de bancos de dados que poderão ser aproveitados.

Até agora não tem havido grande preocupação com o problema de recuperação de falhas em hipertextos devido principalmente a inexistência de restrições de integridade mais completas. Com a implementação de tipos de dados e a definição que norteia a existência e o relacionamento entre estes tipos, fica evidente a necessidade de um mecanismo que restaure a integridade do hiperdocumento quando da ocorrência de uma pane qualquer no sistema.

IV - CONCLUSÕES

Neste artigo procuramos demonstrar a necessidade de discussão de aspectos relativos a utilização de sistemas de gerenciamento de bancos de dados, convencionais ou não, no suporte à arquitetura de sistemas de hipertextos. Esta é uma questão que tem grande relevância na definição e utilização de sistemas de hipertextos para um uso mais geral. O assunto se encontra em estudo dentro do projeto HIPERBASE que visa justamente a unificação dos conceitos de bancos de dados e hipertextos dentro do mesmo ambiente.

Na nossa opinião o desenvolvimento de sistemas de hipertextos para uso mais geral não pode dispensar o suporte de um sistema de gerenciamento de bancos de dados, sob o risco de ou aumentar em muito o tempo de desenvolvimento. Por outro lado, é preciso que os SGBDs ofereçam muito mais flexibilidade do que a disponível nos sistemas atuais, a fim de possibilitar um melhor atendimento aos requisitos diferenciados de sistemas de hipertextos. Aliás, esta reivindicação existe também para outras classes de sistemas não convencionais, como CAD, sistemas de tratamento de imagens, etc. Assim, acreditamos que a linha de orientação a objetos ofereça soluções para estes problemas.

A motivação principal do estudo que está sendo realizado dentro do ambiente PRADES é justamente o desenvolvimento de um sistema de hipertexto para dar suporte ao processo de desenvolvimento de software. O ambiente de hardware atualmente disponível para desenvolvimento do software é um VAX 8810 com o sistema operacional VMS. A

justificativa para esta escolha em detrimento de ambientes tipo PC ou estações de trabalho, é que temos como objetivo desenvolver a ferramenta no mesmo ambiente em que o sistema de informação será implantado.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Borges, M.R.S. Uma análise realista da utilização do enfoque da prototipagem rápida no desenvolvimento de sistemas de informação. Anais do XIII Conferência Latino-americana de Informática, Bogotá, Colômbia, Novembro 1987.

[2] Borges, M.R.S. Rapid prototyping and 4GLs: The change in developing methods of information systems. Anais do IV International Conference: Computers in Institutions of Education, Mexico City, Mexico, Novembro 1988.

[3] Borges, M.R.S. Hipertextos: A próxima revolução no processo de desenvolvimento de sistemas, Congresso Nacional da SUCESU '91, São Paulo, Setembro 1991.

[4] Borges, M.R.S. Combinando bancos de dados e hipertextos: O projeto Hiperbase. Artigo em preparação, 1992.

[5] Bush, V. As we may think. Atlantic Monthly 176 (1), Julho 1945, pp. 101-108.

[6] Conklin, J. Hypertext: An introduction and survey. Computer 20 (9), 1987, pp. 17-41.

[7] Conklin, J & Begeman M. gIBIS: A hypertext tool for exploratory policy discussion. MCC Technical Report STP-082-88, 1988.

[8] Garg, P.K. e Scacchi, W. A hypertext system to manage software life cycle documents. IEEE Software (7) 3, Maio 1990, pp 90-98.

[9] Halasz, F. Reflections on Notecards: Seven Issues for the next generation on Hypermidia Systems, Communications of the ACM, (31) 7, Julho 1988, pp 836-852.

[10] Halasz, F. Seven Issues Revisited. Keynote Talk, Hypertext '91, San Antonio, Texas, Dezembro 1991.

[11] Anais da Conferência Hypertext '89. Pittsburgh, PA, Novembro 1989.

[12] Anais da Conferência Hypertext '91. San Antonio, Texas, Dezembro 1991.

[13] Lima, M.J.D. Hipertextos e suas aplicações. Projeto fim de curso, Departamento de Computação IM/UFRJ, Março 1989.

[14] Meira, S. et al. Hipertexto: O projeto do sistema H. Anais do II Simpósio Brasileiro de Engenharia de Software, Recife, Outubro 1989, pp. 152-170.

[15] Meyrowitz, N. Hypertext - Does it reduce cholesterol, too?, Palestra de abertura do Hypertext '89, Novembro 1989, Pittsburgh, USA.

[16] Reis, A., Borges, M. e Mendes, S. Um modelo híbrido de frames e hipertexto, SBIA'91, Simpósio Brasileiro de Inteligência Artificial, Brasília, DF, 1991.

[17] Schultz, H. e Streitz, N. Hyperbase: A hypermedia engine based on a relational database management system, European Conference on Hypertext '90, Paris, 1990.

[18] Tompa, F.Wm e Raymond, D.R. Database methods for hipertext. Course Notes of Hypertext '91, San Antonio, Texas, Dezembro 1991.

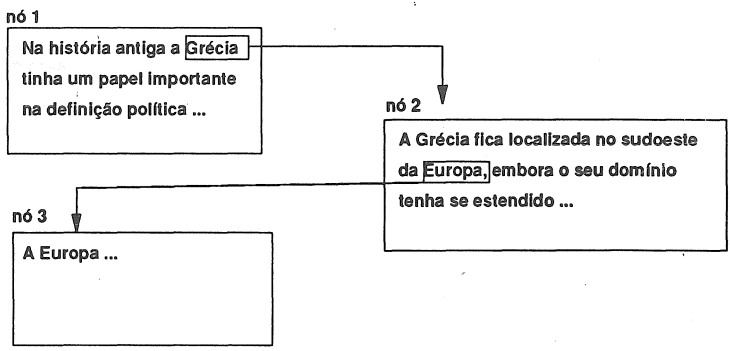


Figura 1 - Exemplo de nós, ligações e botões em um hipertexto

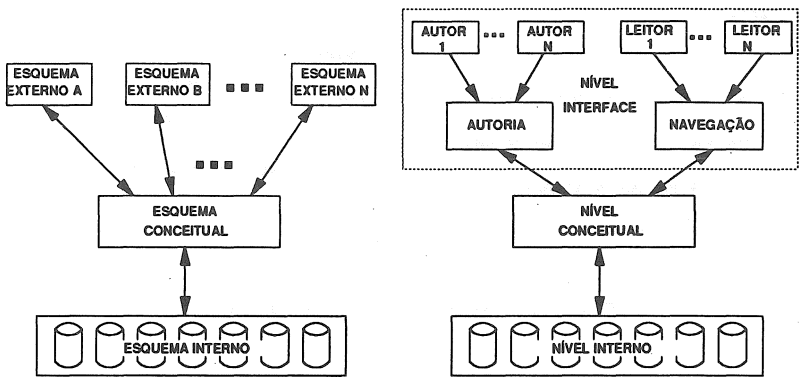


Figura 2 - Arquitetura em 3 níveis de um sistema de hipertexto

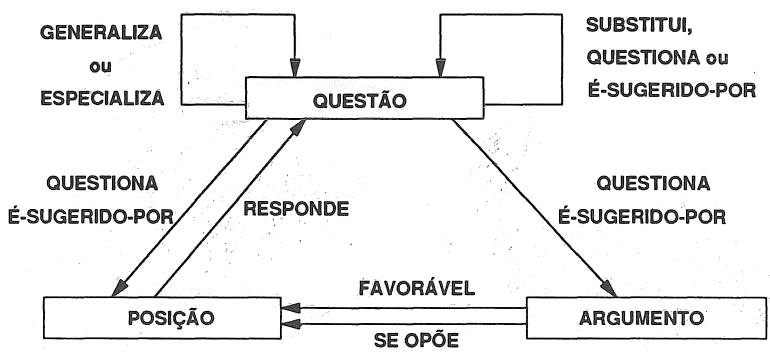


Figura 3 - Tipos de nós e ligações no sistema Gibis

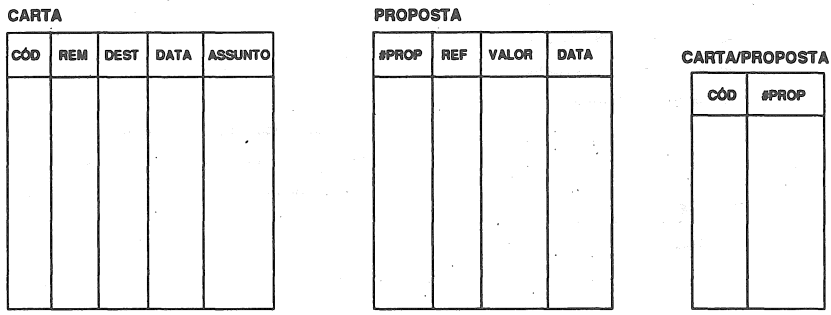


Figura 4a - Esquema relacional do banco de dados CARTA/PROPOSTA

CARTA 736/90

EM RESPOSTA A SUA CARTA ENVIADA NO DIA 14/11 ESCLARECEMOS QUE A NOSSA PROPOSTA SEGUE O MESMO PADRÃO DA PROPOSTA 075G/0690 QUANTO AO ITEM TEMPO DE GARANTIA. ESTA INFORMAÇÃO ALIÁS, JÁ TINHA SIDO REITERADA NA NOSSA CORRESPONDENCIA #567/90 DO DIA 22/11/90.

RELACIONAMENTOS OBSERVADOS:

CARTA -> RESPONDE -> CARTA RECEBIDA
CARTA -> REFERE -> OUTRA PROPOSTA
CARTA -> REFERE -> CARTA ENVIADA

Figura 4b - Instância da entidade CARTA

CARTA

CÓD	736/90	DATA	27/11/90
REM	ANTONIO DE MORAES		
DEST	FAPERJ - EDUARDO PENNA FRANCA		
ASSUNTO	COMPRA DE REGULADORES DE VOLTAGEM		
CONTEÚDO:			
EM RESPOSTA A SUA CARTA ENVIADA NO DIA 14/11 ESCLARECEMOS QUE A NOSSA PROPOSTA SEGUE O MESMO PADRÃO DA PROPOSTA 075G/0690 QUANTO AO ITEM TEMPO DE GARANTIA. ESTA INFORMAÇÃO ALIÁS, JÁ TINHA SIDO REITERADA NA NOSSA CORRESPONDENCIA #567/90 DO DIA 22/11/90.			

Figura 4c - Entidade CARTA modelada conceitualmente

Specifications from Multiple Viewpoints: The Temporal-Causal Way

Jaelson Freire Brelaz de Castro *

Departamento de Informática – UFPE

Caixa Postal 7851– 50732 Recife– PE

Brazil

jbc@di.ufpe.br

Abstract

In this paper we argue that a single representation scheme is often not enough to capture the various features of system behaviour. Instead, multiple viewpoints should be used to partition the domain of information.

We present a technique that supports the specification of systems from three points of view: informal, causal and temporal-causal. The *Informal View* describes in natural language and graphically the requirements of the component. It is based on the *Conic Environment* which provides a language-based approach to the building of distributed system. The *Causal View* relies on enhanced Petri Nets to describe internal structure, distributed control and safety requirements. The third view makes use of temporal-causal logic to describe temporal features of the components such as liveness requirements. Previous causal properties are preserved. Temporal-causal logic is the result of the integration of two well-known formalisms, namely Petri Nets and Temporal Logic.

Keywords

Distributed Systems, Formal Methods, Specification.

*I gratefully acknowledge CNPq, under grant 301052/91-3(NV) for its financial support. I also acknowledge the usefull discussions with Jeff Kramer at Imperial College.

1 Introduction

Our primary goal is to develop a framework to support specification, i.e. modeling and analysis, of concurrent and distributed systems. A *system* will be composed of separate, interacting *components*, possibly highly independent of each other. Components will have well-defined interfaces and their local state that changes over time. A system will be described and managed in terms of their configuration [5].

It is well known that despite the considerable research work on automated software construction and formal methods, precise specifications for complex systems are still proving to be extremely difficult to develop. This lack of impact on the *real world* may be attributed to the inadequacy of existing techniques. Often, they are suitable for representing some limited aspects of the systems, but are unable to characterize many other features. Therefore, a single representation scheme is often not enough to capture all the features of the behaviour of a system.

For example, the standard Petri Net technique provides a graphical notation that has direct and immediate appeal for small examples. It is very suitable for the initial phases of the description, when one is still working out the details and is mainly interested in capturing the flow of control. The possibility of using several levels of abstraction is clearly an advantage. But, for larger systems it usually leads to large and unstructured graphs that are difficult to understand. Besides, temporal properties are not directly stated within this framework.

On the other hand, temporal logics have proved to be good at modeling the reactive aspect of systems and expressing temporal properties. Yet, large specifications written directly in temporal logics are unstructured collections of formulae from which it is difficult to produce or even extract the meaning.

We argue that a single representation scheme is often not enough to capture the various features of system behaviour. Instead, multiple viewpoints should be used to partition the domain of information. Its success rests upon the selection of appropriate representation schemes, the careful definition of the relations between them and the process by which such specifications are built within those representation schemes.

Indeed, the major feature of research in software engineering and formal methods in recent years has been the trend towards combinations and integrations of different approaches [8, 9, 3].

In this paper we show how multiple representation schemes, such as temporal logics and Petri Nets, can be used to describe the behaviour of systems. This temporal-causal framework will contribute towards the provision of a more effective basis for software development, enabling system specification from multiple points of view. The main advantages are the ability to express explicitly, clearly and compactly the causal and temporal relationship between the events of the system. Formal reasoning is also supported, i.e local and global properties are derived [2].

We propose the specification of systems from three points of view: informal, causal and temporal-causal. The *Informal View* describes in natural language and graphically the requirements of the component. It is based on the *Conic Environment* that provides a language-based approach to the building of distributed system. The *Causal View* relies on enhanced Petri Nets to describe internal structure, distributed control and safety requirements. The third view makes use of temporal-causal logic to describe temporal features of the components such as liveness requirements. Previous causal properties are preserved.

In [4] we show how Petri Nets and Temporal Logic can be integrated. In essence, the basic Petri Net model is enhanced (to include types, guarded transitions and the annotation of places with a logical scope) and given a logical proof-theoretic characterization. This logic is then merged with standard temporal logic. The resulting logic is called *Temporal-Causal Logic* [1].

A methodology for the temporal-causal specification of systems is briefly discussed in **Section 2**. It shows how one can start from an informal description and end up with a temporal-causal specification. **Section 3** demonstrates the viability of the specification technique. The components of the patient monitoring system are described in detail. **Section 4** summarizes the discussions and concludes the paper.

2 Methodology

Given a certain design (possibly obtained with a constructive design method [6]), which identifies a collection of components as well as their structure, our job is to specify the individual components of the systems as well as the structural configuration of the system. The first task consists of:

- Identification of the basic components that will the basis for the system;

Having done that, one should now concentrate on the specification of the individual components of the system. As discussed above, a component specification consists of three complementary views. The informal one is the starting point of the process. The objective is to recognize the interface ports of the components, as well as messages that are going to flow through. A rectangle with ports should be drawn to represent the component. If possible, message types should be associated with the ports. A description in natural language (e.g. English) should be attached to the graphical description. Since we are going to concentrate on events and their effects, it should list all the events associated with the component. Observe that each "port" has a communication event associated with it. The remaining events are "internal" ones, i.e. related to the control flow of the activities. For each event there should be a rule describing what are the enabling conditions and what is the effect of its occurrence.

Having produced the informal description of the component, the next step is to provide the *causal view.* As explained above, it will correspond to an enhanced net. At a macro level one could portray a component as one transition whose input places are associated with the entryports of the informal view, and output places corresponds to the exitports. These places are typed according to the messages that they suppose to carry.

At a more detailed level, the initial transition should be refined into a rectangle with has the same input/output typed places. The interior of the rectangle may contain several other transitions (one for each rule identified in the informal view) and places. The idea is to be able to represent graphically the flow of control of activities that the component may perform. Transition and places will correspond to circle and bars respectively. Conditional transitions (those which occurrences depend upon the holding of a guard) are depict as hatched bars. A flow relation captures the relationship between the conditions and transition. Scopes also may be associated with transitions and inhibitor arcs are also allowed. Recall that all the entities present in the enhanced net diagram are typed. We shall adopt the convention of underlining condition predicates and representing inhibitor arcs as dashed lines which end in black dots (not arrow). Besides, an outbound arrow associated with a transition, may have underneath it a "scope", i.e. a formula of the Enhanced net language. It is expected that several versions of the enhanced net may be required until the specifier is "convinced" that it indeed captures the required structural behaviour of the component. Only then, the enhanced net description should be coded-up into causal (the effect of transition occurrence) and safety (when a transition is enabled to fire) axioms.

The last stage is the characterization of the further temporal properties of the component, i.e. liveness. For instance, it will be possible to capture when a transition must fire. These requirements are expressed as temporal-causal logic axioms. Any other property that the specifier feels that may be required to be expressed, also should be written as additional logical axioms. The result is a set of axioms, which represent causality aspects as well as temporal properties.

Having defined the set of logical axioms that characterize the behaviour of the component, one should then make use of the logical machinery available to prove properties of the specification. It is through these logical manipulations that confidence in the specification is gained.

Systems are constructed as a composition of the various individual components. In the remaining of this paper we shall concentrate on the component specification, leaving system specification for [4].

3 An example: The Patient Monitoring System

The example to be used to illustrate the temporal-causal technique is the patient monitoring system (PMS) [7]. The intensive care ward in a hospital consists of a number of factors, such as pulse, temperature and blood pressure. For each patient the current readings can be displayed both at the bedside and at the nurse unit. If any of the factor readings of a patient is outside the preset limits, then an alarm is sent to the central nurse station. The ward system will be composed of two sorts of components: patient and nurse. Bellow we specify the **Patient Component**.

3.1 Informal View

Three interface ports are required: alarm, bed req and bed reply. Exitport alarm conveys messages which carry alarm information (i.e the status of each sensor attached to the patient). Entryport bed req receives signal messages requesting the latest bed readings. Exitport bed reply relays messages with the most updated patient readings.

The next task is the identification of the basic events of the component. Recall that for each port there should be one event. Therefore, let us denote them as "**send** bed.alarms **to** alarm", "**receive** signal **from** bed req" and "**send** bed **to** bed reply". Three internal events are also necessary, "**new reading**(data)", "**Alert nurse**" and "**check reading**". The first one models the readings of the sensors attached to the

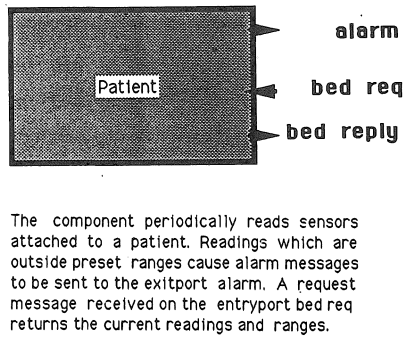


Figure 1: Patient Component: Informal View

patient, the second models the decision to alert the nurse (conditional on problems being detected with the latest patient readings) and the last event indicates that a new reading of the sensors should be performed (conditional on the last readings being normal). Last, the behaviour of the patient component could be informally described as a set of rules, one for each event.

- **R1 Receiving a request for patient information;** a patient is enabled to receive a signal requesting the latest patient readings only if he is in "waiting for new request" condition. The reception of the signal causes the patient to prepare a message (with the latest readings) to be sent.
- **R2 Sending the latest patient readings;** a patient is enabled to send a message with the latest readings only if the appropriate message has been prepared. The sending of the message causes the patient to return to the condition of "waiting for a new request".
- **R3 Performing a new reading of the sensors;** a patient component is enabled to perform a new reading of the sensors attached to a patient only if it is in "read sensors" condition. The reading of the sensors causes a condition where the new "data" acquired must be tested. A variable (bed) is updated with the most recent patient readings and sensors are checked to detect any abnormality. If a problem is detected, then a predicate (inwarning) must be set
- **R4 Alerting the nurse;** a patient is enabled to alert the nurse only if the latest readings have been tested and provided that an abnormality had been detected (predicate inwarning holds). Alerting a nurse causes an alarm message to be prepared to be sent.
- **R5 Sending an alarm message;** a patient is enabled to send an alarm message only if the appropriate alarm message has been prepared. Sending it causes the patient to return

to the condition of "read sensors".

- **R6 Check again the sensors** a patient is enabled to check again the sensors only if the latest readings have been tested and provided that there were no abnormalities (predicate inwarning does not hold). This event causes the condition "read sensors" again to hold.

3.2 Causal View

Based on the informal view given above, the following enhanced Petri Net of Figure 2 can be obtained. Observe that there are two concurrent cycles of activities. The top one deals with the generation of alarm messages. The bottom one addresses the request and reply of patient readings.

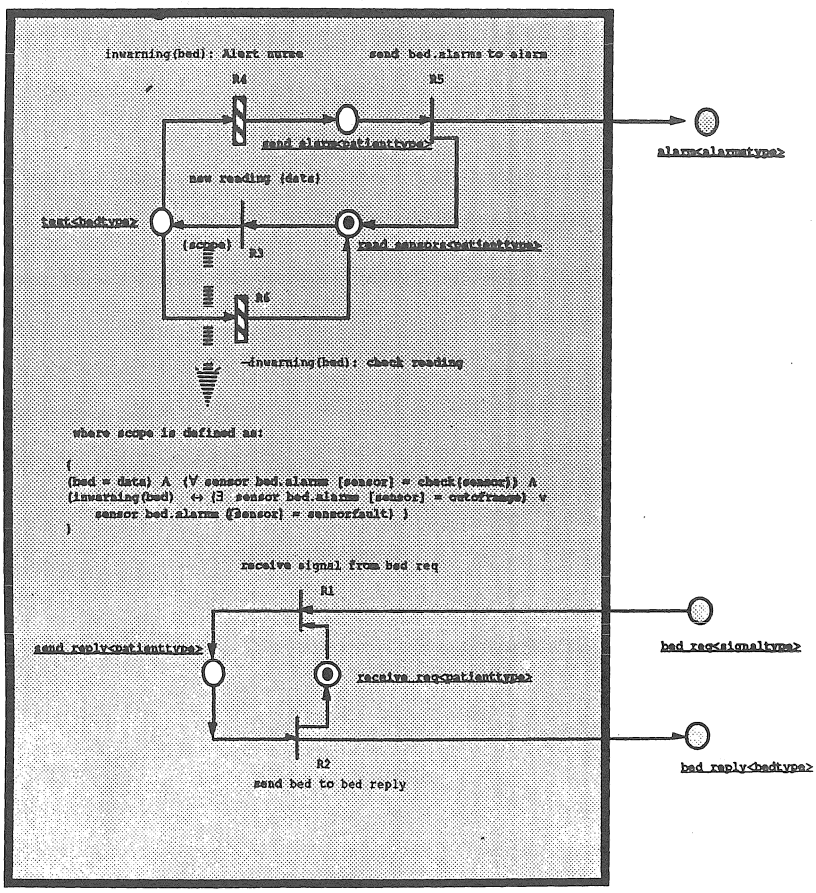


Figure 2: Patient Component: Enhanced Net Description

3.3 Patient Temporal-Causal View

A component specification will consist of a component-id, data-types definition, port-declaration, local-definitions and behaviour. In this example the component-id will be "Patient" and data-types are imported from the definition module "*monmsg*". Among the local-definitions there are variables, condition predicates (for flow control purpose), (normal) predicates and events. The **Behavior** is given in terms of two views: Causal (see Figure 2) and Temporal-Causal presented below. The Temporal-Causal View has four segments: **Causal Properties**, **Safety Properties**, **Liveness Properties** and **Initialization**. The first two are based on the enhanced net description. Each transition in the causal view will correspond to an event in the temporal-causal view, which has a causal and safety property associated with it. One could envisage a scenario in which these two segments are obtained automatically from the causal view with the help of a "translator tool". The **Initialization** segment could possibly be obtained from the enhanced net. It will correspond to the initial "*case*" as well as the setting of the initial values of other logical entities such as predicates.

Liveness properties are unique to the temporal-causal view. They are meant to express when an event must or has to occur. For instance, if a component receives a request for its latest readings (condition predicate bed req(signal) it is committed eventually to reply (occurrence of event "**send** bed **to** bed reply"). Also, if some patient readings are out of range (predicate inwarning(p) holds), then an alarm message will be sent out (occurrence of event "**send** bed.alarms **to** alarm").

Patient Specification

Component Patient;
import monmsg: bedtype, signaltype, alarmstype, patienttype;
exitport alarm: alarmstype;
entryport bed req: signaltype;
exitport bed reply: bedtype;
Variables
bed, data: ⟨bedtype⟩;
p: ⟨patienttype⟩;
signal: ⟨signaltype⟩;
Condition Predicates
bed reply, test: ⟨bedtype⟩;
alarm: ⟨alarmstype⟩;
bed req: ⟨signaltype⟩;
send reply, receive req, read sensors, send alarm: ⟨ patienttype ⟩;
Predicate

inwarning: ⟨bedtype ⟩;

Events

receive *signal* **from** *bed req*: ⟨ signaltype, signaltype ⟩;	(to be used with **R1**)
send *bed* **to** *bed reply*: ⟨ bedtype, bedtype⟩;	(to be used with **R2**)
new reading(data) : ⟨bedtype⟩;	(to be used with **R3**)
Alert nurse: ⟨patientype⟩;	(to be used with **R4**)
send *bed.alarms* **to alarm**: ⟨alarmstype, alarmstype⟩;	(to be used with **R5**)
check reading: ⟨patienttype⟩;	(to be used with **R6**)

Behaviour

Causal View (see Figure 2);

Temporal-Causal View

Causal Properties

P 1 $([\ receive\ \ signal\ \ from\ \ bed\ \ req\ \rangle \overbrace{\underline{send\ \ reply(p)}}^{post-condition} \wedge \overbrace{\underline{\neg bed\ \ req(signal) \wedge \neg receive\ \ req(p)}}^{resetting})$; *(from* **R1***)*

P 2 $([\ send\ \ bed\ \ to\ \ bed\ \ reply\ \rangle \overbrace{\underline{bed\ \ reply(bed) \wedge receive\ \ req(p)}}^{post-condition} \wedge \overbrace{\underline{\neg send\ \ reply(p)}}^{resetting})$; *(from* **R2***)*

P 3 $([\ new\ \ reading(data)\ \rangle\ \overbrace{\underline{test(data)}}^{post-condition}\ \wedge\{\overbrace{(bed = data)}^{scope.1} \wedge \overbrace{(\forall sensor\ bed.alarms[sensor] = check(sensor))}^{scope.2} \wedge$
$\overbrace{(inwarning(bed) \leftrightarrow (\exists\ sensor\ bed.alarms[sensor] = outofrange) \vee (\exists\ sensor\ bed.alarms[sensor] = sensorfault))}^{scope.3}\}) \wedge$
$\overbrace{\underline{\neg read\ \ sensors(p)}}^{resetting}$; *(from* **R3***)*

P 4 $([\ inwarning(bed) : Alert\ \ nurse\ \rangle \overbrace{\underline{send\ \ alarms(p)}}^{post-condition} \wedge \overbrace{\underline{\neg test\ \ (data)}}^{resetting})$; *(from* **R4***)*

P 5 $([\ send\ \ bed.alarms\ \ to\ \ alarm\ \rangle \overbrace{\underline{alarm(bed.alarms \wedge read\ \ sensors(p))}}^{post-condition} \wedge \overbrace{\underline{\neg send\ \ alarm(p)}}^{resetting})$; *(from* **R5***)*

P 6 $([\ \neg inwarning(bed) :\ \ Read\ \ again\ \rangle \overbrace{\underline{read\ \ sensors(p)}}^{post-condition} \wedge \overbrace{\underline{\neg test(data)}}^{resetting})$; *(from* **R6***)*

Safety Properties

P 7 $(\overbrace{\underline{bed\ \ req(signal) \wedge receive\ \ req(p)}}^{precondition} \rightarrow \bigcirc Occ(receive\ \ signal\ \ from\ \ bed\ \ req))$; *(from* **R1***)*

P 8 $(\overbrace{\underline{send\ \ reply(p)}}^{precondition} \rightarrow (\bigcirc Occ(send\ \ bed\ \ to\ \ bed\ \ reply))$; *(from* **R2***)*

P 9 $(\overbrace{\underline{read\ \ sensors(p)}}^{precondition} \rightarrow (\bigcirc Occ(new\ \ reading(data))$; *(from* **R3***)*

P 10 $(\overbrace{\underline{send\ \ alarm(p)}}^{precondition} \rightarrow (\bigcirc Occ(send\ \ bed.alarms\ \ to\ \ alarm))$; *(from* **R5***)*

P 11 ($\overbrace{\underline{test(data)}}^{precondition} \wedge inwarning(bed)) \rightarrow (\bigcirc Occ(Alert\ \ nurse))$; *(from* **R4***)*

P 12 ($\overbrace{\underline{test(data)}}^{precondition} \wedge \neg inwarning(bed)) \rightarrow \bigcirc(Occ(check\ \ reading))$; *(from* **R6***)*

Liveness Properties

P 13 $\underline{bed\ \ req(signal)} \rightarrow \Diamond Occ(send\ \ bed\ \ to\ \ bed\ \ reply)$

P 14 $inwarning(bed) \rightarrow \Diamond\ Occ(send\ \ bed.alarms\ \ to\ \ alarm)$

Initialization

P 15 $\underline{read\ \ sensors(p)} = true \wedge \underline{receive\ \ req(p)} = true \wedge inwarning(p) = false$

end.

The above example demonstrates how the technique can be used to specify individual components of the system. The next step would be the specification of the system as an interconnected set of component instances described by a configuration specification (see [4]). System requirements will be satisfied by the composition of the component specifications.

4 Conclusion and Remarks

We argued that a single representation scheme is often not enough to capture the various features of system behaviour. Instead, multiple viewpoints should be used to partition the domain of information. Its success rests upon the selection of appropriate representation schemes, the careful definition of the relations between them and the process by which such specifications are built within those representation schemes.

We presented a technique that supports the specification of systems from three points of view: informal, causal and temporal-causal. The *Informal View* describes in natural language and graphically the requirements of the component. It is based on the *Conic Environment* which provides a language-based approach to the building of distributed system. The *Causal View* relies on enhanced Petri Nets to describe internal structure, distributed control and safety requirements. The third view makes use of temporal-causal logic to describe temporal features of the components such as liveness requirements. Previous causal properties are preserved. Temporal-causal logic is the result of the integration of two well-known formalisms, namely Petri Nets and Temporal Logic.

Because we concentrated on causal and temporal aspects of system specification, the natural representation schemes were Petri Net and Temporal Logic. The temporal-causal logic was used to define the relations between the techniques.

We then proceeded to give some guidance on how to obtain temporal-causal specification of systems and described the patient component of the patient monitoring system.

5 BIBLIOGRAPHY

References

[1] J. Castro. Temporal-Causal Logic: A marriage of convenience. Technical Report 91/4, Imperial College of Science and Technology, Department of Computing, March 1991.

[2] J. Castro. Reasoning about distributed system specification: The temporal-causal way. In *Proceedings of X Brazilian Symposium on Computer Networks (X SBRC)*, April 1992. Recife, Brazil.

[3] J. Castro and J. Kramer. Temporal-Causal System Specifications. In *Proceedings of IEEE International Conference on Computer Systems and Software Engineering (CompEuro90)*, pages 210–217, May 1990. Tel-Aviv, Israel.

[4] Jaelson F. B. Castro. *Distributed System Specification Using A Temporal-Causal Framework.* PhD thesis, University of London, October 1990. Imperial College of Science, Technology and Medicine, Department of Computing.

[5] J. Kramer. Configuration Programming - A Framework for the Development of Distributable Systems. In *Proceedings of IEEE International Conference on Computer systems and software Engineering (CompEuro90)*, pages 374–384. IEEE Computer Society Press, May 1990.

[6] J. Kramer, J. Magee, and A. Finkelstein. A Constructive Approach to the Design of Distributed Systems. In *Proceedings 10th International Conference on Distributed Computing Systems (ICDCS)*, pages 580–587, Paris, France, June 1990.

[7] W. Myers, G. Myers, and L. Constantine. Structured design. *IBM System Journal*, 13-(12):115–139, 1974.

[8] Wolfgang Reisig. Towards a temporal logic for causality and choice in distributed systems. In J.W Bakker, W. P. de Roever, and G. Rozemberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 603–627. Springer-Verlag, 1989. LNCS 354.

[9] Ichiro Suzuki and HarnGdar Lu. Temporal petri nets and their application to modeling and analysis of a handshake daisy chain arbiter. *IEEE Transactions on Computers*, 38(5), May 1989.

ADQUISICION DE NUEVO CONOCIMIENTO POR COMPARACION Y CONSTRUCCION DE LA JERARQUIA DE CLASES FORMALES

Henri Briand
IRESTE-MIAGE, Université de Nantes (Francia)
José A. Brugos
Departamento de Matemáticas, Universidad de Oviedo (España)
Covadonga Fernández
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería del software, Universidad Politécnica de Madrid (España)
Concepción Pérez-Llera
Departamento de Matemáticas, Universidad de Oviedo (España)
Rafael Portaencasa Baeza
Departamento de Lenguajes y Sistemas Informáticas e Ingeniería del software, Universidad Politécnica de Madrid (España)

Resumen

En este trabajo se construye a través de un proceso basado en comparaciones sobre tripletas de tipos de objetos obtenidos a partir de los hechos elementales (objetos y características) que establece el experto, el primer nivel de la jerarquía de Generalización /Especialización, que denominamos Jerarquía de Clases Formales. Este nivel clasifica los tipos de objeto del Universo del Discurso y genera una partición de las características obligatorias de los tipos de objeto en clases de equivalencia por niveles.

Palabras Clave: Jerarquía de Generalización/Especialización, Característica Obligatoria.

1.- Introducción

La utilización de S.G.B.D. orientadas a objeto supone un importante avance pero no puede ser un fin en sí misma. La etapa siguiente de la evolución de los S.G.B.D. consiste en unificar las Bases de Datos y las Bases de Conocimiento en un mismo sistema. Tales S.G.B.C. han de ser adaptados tanto para gestionar datos clásicos como los conocimientos u objetos complejos [9].

Se han realizado numerosos esfuerzos en el campo de las bases de datos en la línea de mejorarlas semánticamente para que ofrezcan mayor flexibilidad y sobre todo en la de poder incorporarlas en formalismos del campo de la I.A. como red semántica [4], [5], [2], 'frame' [1] modelo particional [6].

El esquema de representación del conocimiento que utilizamos como base de nuestro formalismo es el modelo O.A.S. (Objeto-Asociación Semántica) [11], muy cercano al Modelo Entidad-Relación Binario [10] pero donde se aporta una nueva visión de la definición convencional de asociación. La asociación la planteamos como una fusión de esquemas de características que independiza el hecho de que un objeto posea una característica del hecho de que ésta tenga asociado un valor, con lo cual también podemos independizar las distintas evaluaciones que pueda tener una característica. Tal planteamiento constituye a nuestro entender un punto de aproximación muy importante de cara a la integración de los modelos conceptuales de B.D. en el campo de la representación del conocimiento.

En la sección 2 se plantean dos conceptos básicos del modelo O.A.S. que son el esquema de características de un tipo de objeto y tipo de asociación de esquemas de características.

En la sección 3 se esboza la forma de obtener, lo que llamamos, tipos de objeto primitivos y sus asociaciones irreducibles, a partir del estudio de los hechos elementales.

En la sección 4 se ofrece el método para adquirir nuevo conocimiento, a partir de la lista de tipos de objeto primitivos, por comparaciones sobre tripletas de tipos de objeto, a la vez que se construye la jerarquía de Clases Formales.

2.- Esquema de Característica y Tipo de Asociación entre Esquemas de Característica.

Un hecho elemental es una aserción simple sobre el Universo del Discurso (U.D.). El U.D. es visto como un conjunto de objetos que cumplen unos predicados. Los hechos elementales son aserciones sobre qué objetos particulares cumplen qué predicados particulares. La clase más simple de hecho elemental afirma que un objeto cumple un predicado unario (juega un papel [10]. Definimos esquema de característica, C(O), por un nombre de predicado unario C (por ej. Tiene_olor) y un tipo de objeto O (por ej. flor), cuya representación gráfica será,

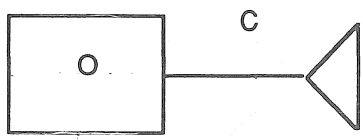


Fig. 1

Una característica c sobre el esquema de característica C(O)es un conjunto finito de objetos c=[Oj/Oj ∈ c, Oj ∈ O] que tienen un cierto significado definido por el cumplimiento del predicado C, por ejemplo c=[rosa, clavel, jacinto,...].

Una tupla de característica es una instanciación de la característica sobre un objeto Oj particular, por ejemplo Tiene_olor (rosa).

Según la representación gráfica al predicado C le damos la dimensión del papel del modelo E-R (papel E-R), aunque este modelo asocia el concepto de papel a la función que los objetos juegan dentro de una asociación, con lo que aquí se le da un sentido más general cercano a la línea del modelo NIAM (papel NIAM); y esto, lo diferenciamos manejando, en nuestro modelo, los dos conceptos: esquema de característica y papel E-R.

Un esquema de característica C(O) puede ser obligatorio u opcional, dependiendo de que a todos los objetos, registrados en la B.C., del tipo de objeto O les exijamos su cumplimiento o no.

Una tupla de característica puede tener una evaluación, a través de una asociación con otra tupla de característica correspondiente a otro esquema de característica.

La diferencia entre esquema de característica y papel E-R o NIAM, radica en que el esquema de característica es interpretado sobre la población del tipo de objeto, y el papel E-R y el papel NIAM, en el caso de una asociación, son interpretados sobre la población de la asociación donde participa ese tipo de objeto a través de un esquema de

característica. Con respecto a la obligatoriedad del esquema de característica/papel, la relación entre ellos es:

"Si un esquema de característica es opcional, entonces el papel es opcional pero no a la inversa".

"Si un papel es obligatorio, entonces el esquema de característica es obligatorio, pero no a la inversa"

Por ejemplo, puede ser que el tipo de objeto Flor posea obligatoriamente la característica de Tener_olor, sin embargo eso no significa que el papel Tener_olor sea obligatorio, es decir que se registre obligatoriamente en la B.C. un olor determinado para cada flor, porque puede haber razones para hacerlo opcional como, por ejemplo, no conocer los olores de algunas flores.

La asociación entre tipos de objeto la planteamos como una fusión de esquemas de características, así, por ejemplo, un tipo de asociación binaria entre esquemas de características se define como un conjunto de pares de objetos (i,j) de dos características c i, cj sobre los esquemas de características Ci, Cj respectivamente. Su representación gráfica será:

donde el significado de los componentes de esta figura es:

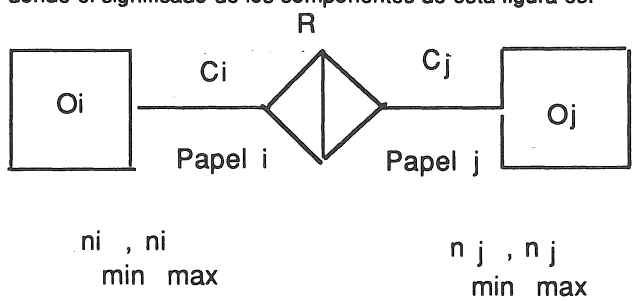


Fig. 2

Oi : Conjunto de objetos sobre el que se define el predicado Ci

Oj : Conjunto de objetos sobre el que se define el predicado Cj

R :Nombre del predicado que cumplen las parejas de objetos asociados (i,j) que pertenecen a las características ci,cj respectivamente; o simplemente un identificador de la asociación.

ni : Cardinalidad del segmento i (número de elementos de R asociados a uno dado de Ci (Oi)) puede ser máxima o mínima.

Ci, papeli : Característica y papel (E.-R) que poseen los objeto de Oi que se asocian.

Cuando la cardinalidad mínima del segmento i es 1 se dice que el esquema de característica es obligatorio, en el caso de que sea o se dice que es opcional.

La diferencia entre Ci y el papeli (E-R o NIAM) radica en que restricción de papel obligatorio es interpretada en términos de la población del tipo de objeto y no en términos de la población de la asociación.

Decimos que R tiene papeles y, que O tiene características. Las características funcionan como papeles de la asociación, cuando se evaluan.

Cj, papelj: Característica y papel (E-R) que poseen los objetos de Oj que se asocien.

Lo interesante de esta definición de asociación, es que aquí, la asociación la planteamos como una evaluación de una caracteristica, con lo que podemos independizar el hecho de que un objeto tenga una característica, como por ejemplo tener sabor, del hecho de que esa característica pueda tener un valor, como por ejemplo sabor dulce, y además podemos independizar las distintas evaluaciones que puede tener una característica. Esto es sumamente interesante de cara a la integración de los modelos conceptuales de B.D. en representación del conocimiento. Lo anterior puede ser resumido en la siguiente figura, donde se exponen las distintas tablas de instancias de los hechos que queremos diferenciar siendo éstos, por una parte, las tuplas de la característica y, por otra las tuplas de la asociación (evaluación):

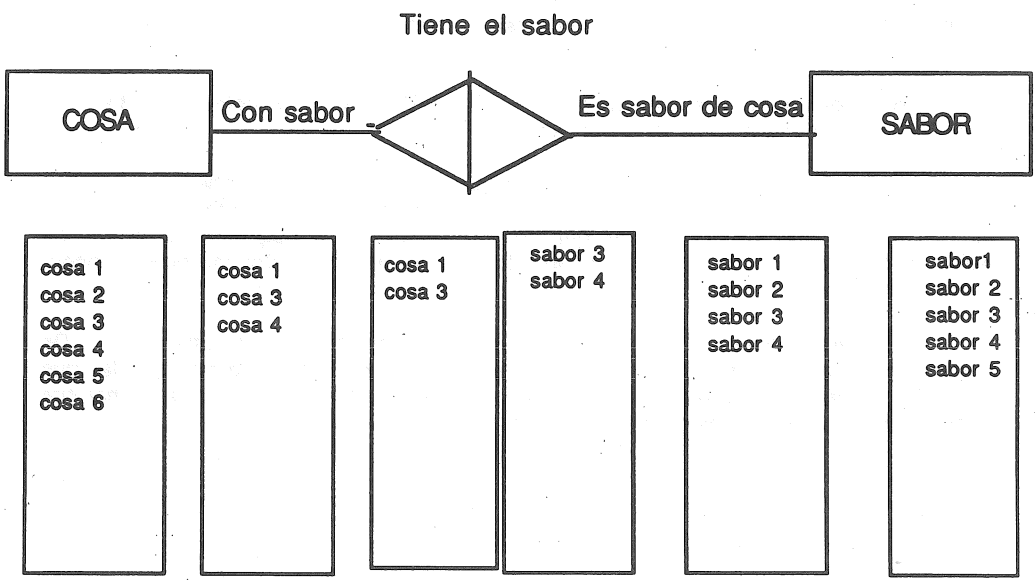


Fig. 3

3. Obtención de los Tipos de Objeto Primitivos y sus asociaciones irreducibles

La fase inicial de adquisición de conocimiento consiste en la recolección de los tipos de objeto y sus características y/o asociaciones, definidas por el experto irreducibles (aquellas que no se pueden descomponer en asociaciones más simples sin pérdidas de semántica), es decir de hechos elementales ligados a sentencias elementales en lenguaje natural, informes, documentos, ejemplos familiares,...

Nuestro interés básico es dividir el U.D., inicialmente de tal forma que cada objeto sea clasificado en exactamente un tipo de objeto. Por eso al seleccionar los tipos de objeto debemos asegurarnos que sean mutuamente exclusivos, esto es, que no tengan instancias en común, combinando tipos de objeto que se solapen en un único tipo de objeto.

Para asegurarnos de que los tipos de asociaciones sean irreducibles nos apoyaremos en métodos formales basados en las restricciones o condiciones de integridad estáticas sobre esos hechos como son las cardinalidades máxima y mínima de un tipo de asociación (de donde se obtienen las Dependencias Funcionales), las Dependencias Multivaluadas y las Dependencias de Unión, y en el proceso de normalización del Modelo Relacional hasta la 5 FN.

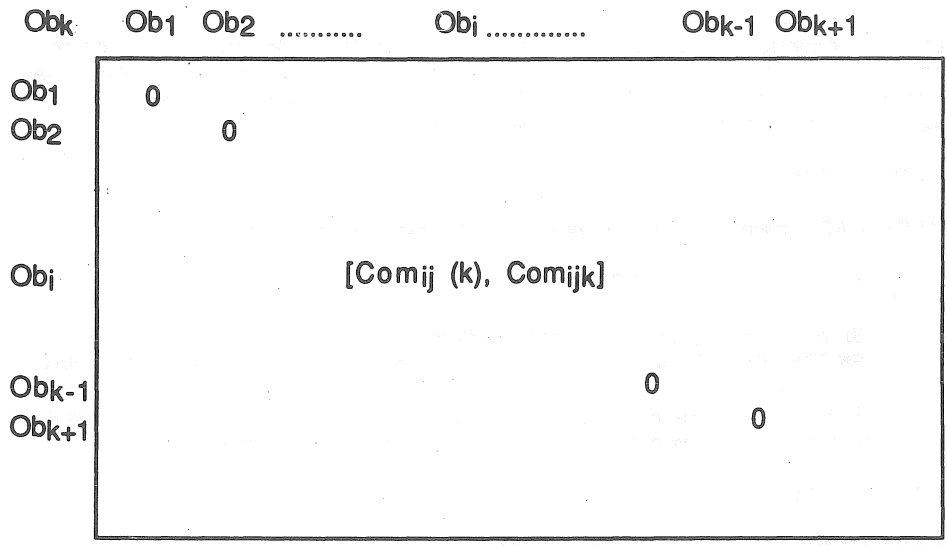
4. Adquisición de nuevo conocimiento y construcción de la Jerarquía de Clases Formales.

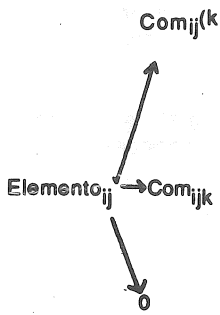
Los tipos de objetos primitivos obtenidos a partir del estudio de los hechos elementales se colocan en la lista LISTA_OBJETOS_PRIMITIVOS, junto con sus características (hechos elementales unarios) y asociaciones (hechos elementales n-arios) iniciales.

Los tipos de objetos de LISTA_OBJETOS_PRIMITIVOS se colocan en tripletas (Obi,Obj,Obk) y el experto ha de establecer:

a) características y/o asociaciones que compartan dos de los objetos (i,j) y que los diferencien del tercero (k). De entre estas características y/o asociaciones (que pueden ser iniciales o nuevas), las que sean obligatorias, se asignarán a un tipo de objeto al que se le suministrará un nombre formal: comij(k), que además puede poseer un nombre natural.

b) características y/o asociaciones que compartan los tipos de objetos i,j,k. De entre estas características y/o asociaciones (que pueden ser iniciales o nuevas), las que sean obligatorias, se asignarán a un tipo de objeto al que se le suministrará un nombre formal: Comijk, que además puede poseer un nombre natural





Tipo de objeto que posee un conjunto de esquemas de caracterísitcas y/o asociaciones obligatorias (iniciales o nuevas) comunes a los tipos de objetos i, j que no comparten con el tipo de objeto k.

Tipo de objeto que posee un conjunto de esquemas de características y/o asociaciones obligatorias (iniciales o nuevas) comunes a los tipos de objetos i,j,k

Los tipos de objeto i,j no comparten caracter´ìsticas y/o asociaciones obligatorias.

Fig. 4

Si Comij(k) en elemij es distinto de $\varnothing$ entonces las características y/o asociaciones que comparten los tipos de objetos i,j (que les diferencien del tipo de objeto k) se eliminan de éstos y se crean relaciones hijo-padre entre ellos y Comij(k).

Si Comijk en elemij es distinto de O entonces las características y/o asociaciones que comparten los tipos de objetos i,j,k se eliminan de éstos y se crean relaciones hijo-padre entre ellos y comijk.

Si se dispone de solamente dos tipos de objeto primitivos es obvia la generación de niveles, pues sólo se solicitarían al experto las características y/o asociaciones que comparten, las cuales se almacenarían en Comij.

En vez de manejar k tablas (una por cada k), donde cada elemento de cada una de las tableas posea dos valores, lo que hacemos es almacenar en elemij una lista de (k-2)*2 elementos.

El propósito de esta técnica es utilizar las características y/o asociaciones que utiliza el experto para discutir los conceptos del dominio del problema y diferenciar objetos, por tanto clasificarlos. El proceso de construcción de la jerarquía clases formarles parte de LISTA_OBJETOS_PRIMITIVOS y está basado en la aplicación del siguiente procedimiento recursivo que genera los niveles de la jerarquía.

GENERA_NIVEL (L: lista_objetos)

PARA CADA pareja de tipos de objetos i,j distintos (j>i) **HACER**

PARA CADA k≠i,j **HACER**

SI los objetos i,j poseen nombres naturales
ENTONCES Generar Comij(k) y Comijk de la forma expresada más arriba.

SI NO Generar Comij(k) y Comijk por simple comparación conjuntista entre los conjuntos de características y/o asociaciones de los objetos i,j,k;

FIN_DE_HACER

PARA CADA elem ij < > lista-vacía **HACER**
Renombrar a nombres únicos los Comij(k) y Comijk en la lista del elemento elemij a partir de las listas de los restantes elementos, en base a la igualdad del conjunto de características y/o asociaciones que posean;

Construir a partir de la lista del elemij el siguiente nivel del grafo eliminando en los objetos i,j,k (en el caso de un Comijk) las características y/o asociaciones que estén en los Com y creando una relación hijo-padre entre ellos y el Com;

FIN_DE_HACER

lista ← unión de las listas de los elementos elemij;

SI lista no está vacía **ENTONCES** ejecutar GENERA_NIVEL (lista)
SI NO acabar;

El resultado es una red, que llamaremos *grafo de clases formales* que se corresponde con una jerarquía de generalización, de la forma:

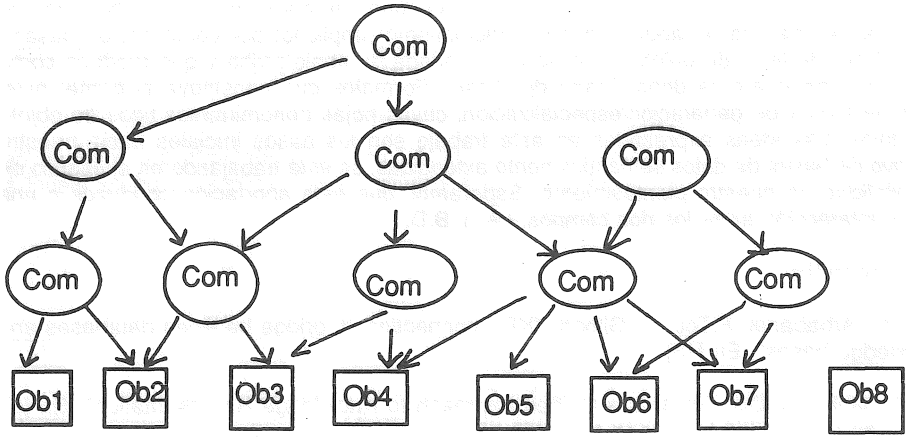


Fig. 5

Las hojas son los tipos de objeto primitivos, y se derivan utilizando el principio de predicabilidad [12], según el cual clases de cosas diferentes tienen clases de predicados distintos, y, de este modo, las cosas pueden ser clasificadas de acuerdo con los predicados que se les aplican. Los predicados pueden estar ligados con las categorías a varios niveles dentro del grafo, y una vez que el predicado está ligado a una categoría se aplica a todas las categorías dominadas por ella.

Para esta red se cumple que los nodos y las hojas son disjuntos dos a dos, en lo que respecta sus conjuntos de asociaciones y/o características obligatorias, es decir, se ha realizado una partición del conjunto global de características y/o asociaciones obligatorias. Este conjunto estará formado por el conjunto de características y/o asociaciones iniciales más las características y/o asociaciones nuevas que han podido ir apareciendo en el proceso anterior basado en tripletas.

Las flechas que llegan a un objeto/clase representan las herencias de características y/o asociaciones que le corresponden.

Conclusiones y futuros trabajos

En este trabajo se plantea la asociación como una fusión de esquemas de características, aportando una nueva visión de la definición convencional de asociación del modelo Entidad-Relación. Tal planteamiento, el cual es básico en el modelo (O.A.S.) sobre el que subyace este concepto, constituye un punto de aproximación importante en lo que respecta a la integración de los modelos conceptuales de B.D. en el campo de la representación del conocimiento.

Desde la concepción anterior, se ofrece una metodología que proporciona al diseñador la manera de adquirir nuevo conocimiento (implícito) por comparación basada en tripletas de tipos de objeto, proceso que procede de abajo/arriba y que produce como resultado una jerarquía denominada de Clases Formales que constituye el primer nivel de la Jerarquía de generación/especialización, cuyas hojas denominamos tipos de objeto primitivos. Las ideas expresadas en este trabajo son los pasos iniciales hacia nuestro objetivo de bases de datos semánticamente extendidas, se está trabajando en el estudio de la viabilidad de nuestro planteamiento. Esperamos que esta aportación conduzca a una mayor interacción entre los dos campos, I.A. y B.D.

Bibliografía

[1] R.M. Arbabanel, F.Tou, Y. Gilbert. "KEE connection: A bridge between databases and knowledge bases". En [13].

[2] Addanki "KL-DB: Towards a Unified Approach to Knowledge Representation". En [8].

[3] M.L. Brodie, J. Mylopoulos (Eds.). "On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies". Springer-Verlag, 1986.

[4] H. Briand. "Un modéle étendu entité-association et sa représentation dans un base de faits d'un systéme expert". En [7].

[5] H. Briand. "Translation from ad Entity-Relationship diagram into an object oriented database". TOOLS, Paris 1990.

[6] C. Fernández Baizán. "Expresión de Dependencias en el Modelo Particional: Análisis de sus propiedades". Actas del IV Congreso de Lenguajes Naturales y Lenguajes Formales, pp. 579-594. Barcelona 1989.

[7] Actas Congreso INFORSID, Tolouse 1985.

[8] R.A.Meersman, A.C. Sernadas (eds.). "Data and Knowledge (DS-2)". North-Holland, 1988.

[9] J. Mylopoulos. "On Knowledge Base Management Systems" en [3]

[10] G.M. Nijssen, T.A. Halpin. "Conceptual Schema and Relational Database Design". Prentice-Hall, 1989.

[11] M.C. Pérez Llera. "Metodología para la obtención del modelo O.A.S.". Tesis Universidad de Oviedo, 1991.

[12] L. Shastri. "Semantic Networks: A formalization". Artificial Intelligence Vol. 39 pp. 283-355, 1989.

[13] M. Yazdani, M. Richer (Eds.) "AI Tolls and Techniques". Norwood 1988

ÑERANDU: Lenguaje de Soporte de Decisiones en la Industria

César Cabello
Sergio Aranda
Mercedes Rivaldi

Laboratorio de Electrónica Digital
Facultad de Ciencias y Tecnología
Universidad Católica "Ntra. Sra. de la Asunción"
Casilla de correo: 1718 Fax: (595) 21 - 31 05 87
E-mail: ccabello@ledip.py, saranda@venus.py
Asunción Paraguay

Resumen

Este trabajo presenta el lenguaje inmerso en un sistema integrado de Producción-Finanzas (*ARANDU*), desarrollado con el objeto de servir como soporte a las decisiones desde la definición de los productos hasta el planeamiento de producción, compras y el manejo financiero; que surgen en la administración de industrias sometidas a variabilidad en la demanda de productos, oferta de materias primas y disponibilidad financiera.

1 Introducción

En los países de producción fundamentalmente agropecuaria, la economía está fuertemente influenciada por los ciclos de la naturaleza; esto impone la necesidad de planificar las actividades productivas, y en particular la industrial, tomando en cuenta las complejas interacciones que resultan de las mencionadas variaciones. A este fin fue desarrollado *Arandú*, nombre del sistema original de Producción - Finanzas a cuya extensión se orienta *Ñerandú*. El objetivo es crear un ambiente *Arandú*, que manteniendo la interactividad y sencillez, pueda ser objeto de programación por parte del usuario.

2 Ambiente de aplicación

Para una adecuada comprersión de *Ñerandú*, es necesaria una descripción, al menos sucinta, de *Arandú* (en su versión original).

Tomando en cuenta las funciones de la labor industrial en que la informática puede brindar mayor apoyo, planificación y control, el ambiente *Arandú* se dividió en los siguiente módulos:

a) Planificación
b) Control de Producción
c) Control Financiero

La lógica de trabajo reside en que el usuario, sobre la base de la información disponible: estructura del producto, procesos realizados sobre el producto y recursos utilizados, estimaciones de ventas por periodos, estimaciones de costo de compra, estimaciones de costo de almacenamiento, estimaciones de costos fijos, estimaciones de disponibilidad de horas hombres y máquinas, y estimaciones de costos de horas hombres y máquinas; elabora un plan o planes alternativos en cuyo cumplimiento se generan productos y consumen recursos. El plan puede ser opcionalmente obtenido en base al soporte de la máquina que en todo su ambiente provee algoritmos para la toma de decisiones fundados en programación matemática, programación dinámica y heurísticas ad hoc, elegibles según el criterio del usuario.

El planeamiento de la producción, de las compras y de las finanzas pueden ser efectuados en un horizonte de *n* periodos, bajo diferentes criterios de optimización. Una vez aceptado un plan es objeto de un *lanzamiento*; a partir de este momento el cumplimiento del plan es objeto de control en los niveles productivos y financieros.

Este esquema de trabajo se ha demostrado eficaz, como lo prueba la experiencia en las empresas que utilizan *Arandú*, pero el análisis de alternativas y planes multiples puede facilitarse significativamente mediante la disponibilidad de recursos informáticos que permitan un control automático del flujo de dichos análisis, sin necesidad de la intervención directa ante cada resultado, de manera similar a como operan las macros en las planillas electrónicas. De esta necesidad nace ÑERANDU.

3 Descripción del Lenguaje

Los comandos definidos en Ñerandú permiten al usuario procesar la información contenida en las tablas definidas en el sistema. Los comandos se agrupan en tres conjuntos :

a) *Algebra relacional*, permiten el manejo eficiente de bases de datos relacionales.
Ejemplos : **CREAR_TABLA**, permite al usuario crear una tabla, ya sea para ser usada como introducción de datos, así como para obtener resultados de salida.

BORRAR_TABLA, permite al usuario eliminar una tabla previamente creada.

b) *Estructuras de control de flujo*, permiten al usuario avanzado crear programas que interactúan con las bases de datos.
Ejemplos: **SI-FINSI**, permite un salto condicional en la ejecución de un grupo de sentencias .

PARA-FINPARA, permite la iteración de un conjunto de sentencias.

c) *Operaciones matemáticas complejas*, incluyen la posibilidad de plantear y resolver un problema de programación lineal.

Ejemplos : **PRG_LINEAL**, soluciona un problema lineal.

CAMBIO_DIS, dada la solución de un problema lineal, permite resolver uno semejante partiendo de la solución anterior, pero con cambios en las restricciones de *disponibilidades*. Puede de esta manera evitarse el replanteo completo y el cálculo desde el comienzo de la solución,

CAMBIO_COST, semejante a la función anterior, pero los cambios permitidos son sobre los costos de las variables de decisión del problema lineal.

El usuario puede utilizar las siguientes macro funciones, que permiten aprovechar toda las prestaciones de *Arandú* :

A - **OPTIM_MEZ**(SemiElaborado, TablaResultado), que permite optimizar la composición de Semi-Elaborados.

B - **ALTER_PROD**(Alternativa, TablaResultado), genera los resultados correspondientes a una alternativa de producción.

C - **PLAN_PROD**(Estimación, Perspectiva, TablaResultado), genera un Plan de Producción, dada una estima y un perspectiva.

D - **PLAN_FINAN**(Estimación, Perspectiva, TablaResultado), genera un Plan Finaciero dada una estima y una perspectiva.

Por razones de consistencia de datos, los comandos definidos no pueden en ningún caso modificar la estructura de las tablas definidas en Arandú.

La descripción formal del lenguaje se presenta siguiendo la notación BNF para definición de sintáxis. Por convención, los términos en **negritas** representan los *símbolos terminales* del lenguaje.

Expresiones Booleanas

ccond -> ccond **.or.** cterm | cterm

cterm -> cterm **.and.** cfactor | cfactor

cfactor -> **.not.** cfactor | (ccond) | **VERDADERO** | **FALSO**

Operadores Matemáticos

op1 -> op1 + op2 | op1 - op2 | op2

op2 -> op2 * factor | op2 / factor | factor

factor -> factor ^ numero

numero -> - num

Números

num -> num digito | num . digito | digito

digito -> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Variables

var -> var alfanum | alfanum

alfanum -> a | b | c | d | . . . | z

Estructuras de Control

A - Estructura PARA

cicloPara -> PARA(inicializaciones ; ccond ; increment) sentencias FINPARA

inicializaciones -> inicializaciones, varInit

increment -> increment , varInit | varInit

varInit -> varInit, var:= op1 | var:= op1

B - Estructura SI

estrucSi -> SI ccond sentencia [SINO sentencia] FINSI

Funciones

stmt -> stmt agregT | stmt borrarT | stmt crearT | stmt nFilT | stmt LinealT | stmt unionT | stmt cicloPara | var:= stmt | stmt estrucSi | stmt cambioDisT | stmt cambioCostT

agregT -> AGREGAR(var , varInit) ;

borrarT -> BORRAR_TABLA(listVar); | BORRAR_TABLA(TODAS);

crearT -> CREAR_TABLA var CON listVar ;

nFilT -> NFILAS(var, ccond) ;

LinealT -> PRGLINEAL([MAX | MIN], var, var, var);

unionT -> UNION(var, var, ccond, [var,...], var);

cambioDisT -> CAMBIO_DIS(var, var, var);

cambioCostT -> CAMBIO_COST(var, var, var);

listVar -> listVar var | var

Inicio

Simbolo de Inicio -> sentencia

sentencia -> sentencia ; stmt | stmt

Símbolos y Convenciones

( ) Indica la lista de argumentos de una función.

[ ] Indica un item o lista opcional.

< > Indica un item que debe ser introducido por el usuario.

, Separador para lista de items.

; Fin de una sentencia.

Operadores y relaciones lógicas

. Indicador de campo.

[ ] Indicador número de fila

.+. Suma de todos los campos de dos tablas. (Alcance de conjunto)
.

.-. Diferencia entre todos los campos de dos tablas.(Alcance de conjunto) .

.*. Producto entre todos los campos de dos tablas.(Alcance de conjunto) .

./. División entre todos los campos de dos tablas.(Alcance de conjunto) .

+ Suma entre dos variables o campos.(Alcance simple) .

- Diferencia entre dos variables o campos. (Alcance simple).

* Producto entre dos variables o campos. (Alcance simple).

/ División entre dos variables o productos. (Alcance simple).

:= Asignación. (Alcance de conjunto o simple, dependiendo de las variables).

== Comparación de igualdad para conjuntos y variables simples.

<= Comparación menor o igual que, para conjuntos y variables simples.

>= Comparación mayor o igual que, para conjuntos y variables ismples.

< Comparación estrictamente menor que, para conjunto y variables simples.

> Comparación estrictamente mayor que, para conjuntos y variables simples.

<> Desigualdad, para conjuntos y variables simples.

.AND. Operación '*Y*' lógica.

.OR. OperaciÛn '*O*" lógica.

.NOT. Negación lógica.

A título ilustrativo se describe en detalle la función **PRGLINEAL** .

PRGLINEAL
Resuelve un programa lineal dado.

Sintáxis

PRGLINEAL(<maxMin>, <funcionObjTabla>, <restricListTablas>, <resultTabla>);

Argumentos

<maxMin> establece el tipo de optimización que se requiere. Así para maximizar un programa lineal se utiliza la palabra clave **MAX**, mientras que se utiliza **MIN** en el caso de minimizar.

<funcionObjTabla> es la tabla o relación en la que se encuentra definida la *función objetivo* del programa lineal dado.

<restricListTablas> es una lista de tablas. Las mismas definen los *requerimientos de recursos* por cada variable de decisión, y *restricciones de disponibilidades* del programa lineal. Cada elemento del vector es una terna formada por, una tabla de requerimientos de recursos, el símbolo <= ó =>, y la tabla de disponibilidades de recursos. Los símbolos de menor que y mayor que indican la relación existente entre las dos tablas anteriores.

<resultTabla> es el nombre de la tabla donde se guarda la *solución del programa lineal* planteado. Esta tabla se crea automáticamente si no existe siguiendo el formato definido en la tabla correspondiente a la función objetivo.

En caso de encontrarse ya definida la tabla, sólo se modifica su formato, si fuere necesario, y su contenido.

Advertencia : Toda información que previamente contenga la tabla solución se pierde.

Descripción

La función **PRGLINEAL** resuelve un problema de programación lineal, planteado en los términos de sus parámetros.

Ejemplo

/ Un fabricante de dos tipos de cerveza, desea saber cuales deben ser los niveles de producción semanal de cerveza clara y oscura que hagan máximo los ingreso por semana. Se sabe que 1000 lts de cerveza clara originan un ingreso semanal de $ 5000, utilizan 3 obreros para ser producidos y su producción cuesta $500. Se sabe también que 1000 lts de cerveza oscura originan un ingreso semanal de $3000, se precisa de 5 obreros y su producción cuesta $200. Se sabe además que la planta tiene un total de 15 obreros y no puede gastar más de $1000 semanales. */*

```
CREAR_TABLA     ingresos_semanales
CON  producto,  caracteres,     15,
     costos,          numérico,  10;
CREAR_TABLA     obreros_por_producto
CON  producto,  caracteres,     15,
     nro_obreros,    numérico,  10;
CREAR_TABLA     disponibilidad_de_obreros
CON  producto,  caracteres,     15,
     tot_obreros,    numérico,  10;
CREAR_TABLA     costos_por_producto
CON  producto,  caracteres,     15,
     costo_unitario, numérico,  10;
CREAR_TABLA     tot_gastos
CON  producto,  caracteres,     15,
     tot_costos,     numérico,  10
```

```
/* Se cargan los datos ya enunciados anteriormente */
AGREGAR(ingresos_semanales,
        "cerveza clara" : 5000,
        "cerveza oscura":1000);
AGREGAR(obreros_por_producto,
        "cerveza clara" : 3,
        "cerveza oscura":5);
. . .
. . .
/* Resolver el programa lineal formulado, guardando el resultado en
la nueva tabla cantidades_optimas */
PRGLINEAL(MAX , ingresos_semanales,
                obreros_por_producto,<= , disponibilidad_de_obreros,
                costos_por_productos, <= , tot_gastos,
                cantidades_optimas ) ;
```

4 Conclusión

Las facilidades que ofrece un lenguaje de soporte de decisiones son significativas respecto a los sistemas de acceso fijos, sin embargo aún queda camino por recorrer en la búsqueda de mayores facilidades para consultas y optimizaciones en sistemas de soporte de decisiones para la industria. Los avances en el procesamiento del lenguaje natural son promisorios para este campo.

Bibliografía

- Aho, Alfred - Sethi, Ravi - Ullman, Jeffrey
  Compilers: Principles, Techniques, and Tools.
  Addison-Wesley Publishing Company.
  1986

- Cabello, César
  Arandú: Soporte de Decisiones en la Industria
  Laboratorio de Electrónica Digital, Universidad Católica
  Asunción - Paraguay
  1991

Elmasri, Ramez - Navathe, Shamkant.
Fundamentals of Database Systems.
The Benjamin/Cummings Publishing Company, Inc.
California - USA
1989

Lee, Ronald M.
Epistemological aspects of Knowledge-based Decision Support Systems.
Processes and Tools para Decision Support
Norht-Holland Publishing Company
1983

Orchard Willian Hays
Advanced Linear - Programming Computing Techniques
McGraw Hill
1968

Prawda, Juan
Métodos y Modelos de Investigación de Operaciones Vol I.
Editorial Limusa.
Ciudad de México - MéxicoNorht-Holland Publishing Company
1989

WRITING TOOLS FOR NON-NATIVE USERS OF ENGLISH

S.M.A.Caldeira[1], M.C.F. De Oliveira[1], N.M.Fontana[2],
C.Y.Nacamatsu[1] and O.N.Oliveira Jr.[3]

[1]Departamento de Computação e Estatística, ICMSC, USP
CP 668, 13560 São Carlos, S.P. - Brazil

[2]Departamento de Letras, Universidade de Caxias do Sul
CP 1352, 95070 Caxias do Sul, R.S. - Brazil

[3]Instituto de Física e Química de São Carlos, USP
CP 369, 13560 São Carlos, S.P. - Brazil

Abstract

*We report on an ongoing project aimed at developing software tools to help non-native users of English in the writing of scientific papers. The environment, named **AMADEUS** (**AM**iable **A**rticle **D**evelopment **E**nvironment for **U**ser **S**upport), is based on an empirical resource which consists in providing linguistic input to the user in the form of expressions and templates stored in a sentences base, and is at an initial exploratory stage, in the sense that we are trying different user-interfaces and environments for developing the tools. One of the versions conceived for the environment is already implemented.*

Keywords: writing tools, non-native users of English, computer assisted writing

I) Introduction

The development of software writing aids requires research efforts on a number of different issues, as writers' needs may vary tremendously depending on their experience in writing, on their expertise on the subject they are writing about, and obviously on their knowledge of the language itself. Though some characteristics such as user-friendliness are always desirable, these tools must

generally be targeted at specific user-groups [1].

We have instituted a project aimed at developing software tools that may help non-native users of English in writing scientific papers. The work is essentially based on an (empirical) resource [2] which consists in providing linguistic input to the user in the form of expressions and templates stored in a sentences base. The work reported here may be considered as exploratory in the sense that we are trying different interfaces and environments for developing the tools. In this paper we address the efficacy of the resource proposed, which was analysed through the monitoring of writing exercises by Brazilian post-graduate students (Section II); and the implementation of the resource in software tools (Section III).

II) Analysis of the Resource from a Linguistic Perspective

The difficulties faced by writers when preparing texts in a foreign language seem to arise due to at least two main factors. The first one is sociolinguistic diversity, as languages differ considerably with respect to phonology, syntax and lexicon, as well as at the pragmatic level [3]. Consequently, different languages use different rhetorical patterns and elements to organize discourse [4]. The second factor which is likely to affect the learner's written outcome has to do with the relationship that holds between comprehension and production: it is now largely accepted that comprehension is beyond production. The combination of these two factors appears to be connected with a phenomenon known as transfer in the linguistic area, whose influence on L2 data has been investigated by several linguists. Here we follow Schachter [5] in her cognitive approach to language transfer. As transfer can often lead to transfer errors, it is our belief that

its negative effects could be prevented, to a certain extent, by providing the non-native writer with expressions from authentic texts from books and papers written by competent native speakers of English to help him/her in organizing their texts more effectively and naturally.

In order to test the efficacy of the resource proposed and also to provide feedback to the process of developing the software tools we monitored writing exercises done by post-graduate students. This was carried out as part of a course on technical writing held in the Physics Department of University of São Paulo in São Carlos. The results are extremely encouraging for using the resource minimized effects of L1 (Portuguese) transfer on L2 (English), and also enabled students to get started in their writing tasks. Indeed, getting over the writer's block has been mentioned as one of the most important problems faced by writers [1].

III) The *AMADEUS* Environment

AMADEUS will comprise composition tools to assist users in the writing of technical articles in English by offering linguistic input in the form of contextualized expressions. It is being developed within the XView (X Window-system-based Visual/Integrated Environment for Workstations)1 a user-interface toolkit to support interactive, graphics-based applications running under the X-Window system in a Sun Workstation. XView features an object-oriented style interface following the OPENLOOK Graphical User-Interface (GUI) specification. We are following a prototyping approach in the development of the environment, and the X-Windows and Emacs-based tools to be discussed shall provide starting points for further developments.

The expressions from the sentences base may be accessed by one of three ways:

1. **Taxonomy** - The expressions are indexed according to a taxonomy which is based on the main topics likely to appear in a paper, and are accessed by selecting an item in the taxonomy.

2. **Communicative Goals** - The search can be performed by selecting a communicative goal (describe, justify, compare, contrast, etc.).

3. **Keywords** - Here the search is carried out by selecting a keyword that appears frequently in texts of a given area. It can also be used for linguistic items which are usually troublesome. For instance, if the phrasal verbs *carry out* and *carry on* are chosen to be keywords, a search into the database will show the correct contextualized uses for these verbs.

Users can be classified according to two main axes, namely their background knowledge of the English language and their experience in writing papers. In order to meet different users' needs we have conceived three versions for the writing environment:

i) **Tutorial** - The whole writing process is monitored by the system; a rigid frame for the organization of the paper is offered, hints are given as to what type of material should be included, and then there is an automatic selection of expressions which guides the user throughout the writing exercise. This automatic selection is driven using expert rules. No acquisition mode for entering new expressions is made available so as to avoid inexperienced writers adding inadequate material. This version is the most demanding in terms of computational development, and has not been implemented yet.

ii) **Support** - This version is similar to the Tutorial except that it offers hints as to how better structure the paper rather than

offering a rigid frame, and allows for acquisition of new material to be entered into the sentences base (customization).

iii) **Reference** - This version requires the least computational development effort and is already implemented. The user consults the database (accessing the expressions by one of the 3 possible ways mentioned above) and selects the expressions on his/her own. The tool can be customized since it includes the **Acquisition Mode** by which new expressions can be entered into the sentences base.

Fig. 1 shows a screendump displaying the main features supported by the already implemented **Reference** version of the environment. Two windows are provided, one as a working area while the other displays expressions from the sentences base. Clicking the option **Main Menu** the user is returned to the menu which displays the options **Composition** and **Acquisition**. The options **Article** and **Keywords** correspond to the modes 1 and 3, respectively, for accessing the expressions. The **Communicative Goals** menu is always displayed on the screen because it is the most general mode of accessing the expressions.

An alternative way of implementing the resource is also being attempted which uses the GNU Emacs text editor, a public domain software package. Emacs was chosen because it provides hypertext facilities, it is customizable and furthermore, it is extensible which means that one can go beyond customization by writing entirely new commands, programs in the Lisp language to be run by the Lisp interpreter provided by Emacs. In our implementation use is being made of a special Emacs function called **Outline-mode**, which has built-in routines that hide and display chunks of text which can be classified hierarchically as items and sub-items of a menu or as normal text. The two windows available in Emacs are used for the text being edited and the database of expressions, and new

routines were added to the Outline-mode in order to allow the transfer of expressions from the sentences base to the text. The facilities already available are similar to the Reference Version of the tool implemented in the XView environment. Using this tool requires users to be familiar with Emacs, by no means a simple word processor to use. However, we are developing a friendlier interface which will allow users to employ the tool with a minimum background knowledge of how Emacs works.

What we had in mind when we decided to investigate an alternative implementation for the resource was to allow for customization and portability of the tool, and also investigate the possibility of introducing some degree of "intelligence" into it. Emacs seemed a natural choice for its flexibility. For instance, texts in Emacs can be generated in the Text Mode (ASCII) or in other modes so as to allow integration with wordprocessors such as TEX. Because Emacs is entirely re-programmable new functions can be added either for the introduction of expert rules or for extending the portability of a given tool. The obvious disadvantage of the Emacs alternative is its poor graphics capability (compared to XView), but at the moment it is not possible to predict whether this disadvantage will be overpowered by its other welcome features.

IV) Concluding Remarks

We should stress that the tools being developed are not aimed at generating text in the more strict sense of the so-called Natural Language Generation (NLG), for the user rather than the system will be ultimately responsible for the quality of the text. But we can, nevertheless, seek help in the techniques usually employed in NLG. In this context, systems [6,7] appeared recently

in the NLG literature which utilize essentially the same strategy discussed here. They are based on statistical analysis of large corpora in order to identify standard linguistic constructions in texts of a given field. It may well be, nevertheless, that the studies we are undertaking, related to language acquisition by a non-native user, may also be of some use to the NLG community.

References

[1] N. Williams, "Writers' Problems and Computer Solutions", CALL Volume 2: Computer Assisted Composition, Intellect, Oxford, pp. 5-25, 1990.

[2] O.N. Oliveira Jr., S.M.A. Caldeira, and N. Fontana, "Chusaurus: A Writing Tool Resource for Non-Native Users of English", in Proceedings of the XI International Conference of The Chilean Computer Science Society, pp. 59-70, 1991.

[3] N. Wolfson, "Perspectives: Sociolinguistics and TESOL" Newbury House Publishers, N.York, 1989.

[4] C. Henner-Stanchina, "From reading to writing acts", in Discourse and learning, Riley,P. (ed.), Longman, Burnt Mill, 1985

[5] J. Schachter, "A New Account of Language Tranfer", in Language Tranfer and Language Learning, Gass, S. & Selinker, L. (eds.), Newbury House Publishers, Roweley, pp. 98-111, 1983.

[6] P.L. Miller and G.D. Rennels, "Prose Generation from Expert System - An Applied Computational Linguistics Approach", AI Magazine, pp. 37-44, Fall 1988.

[7] F.A.Smadja and K.R.McKeown, Automatically Extracting and Representing Collocations for Language Generation, 28th ACL Meeting, Pittsburg, PA, 1990

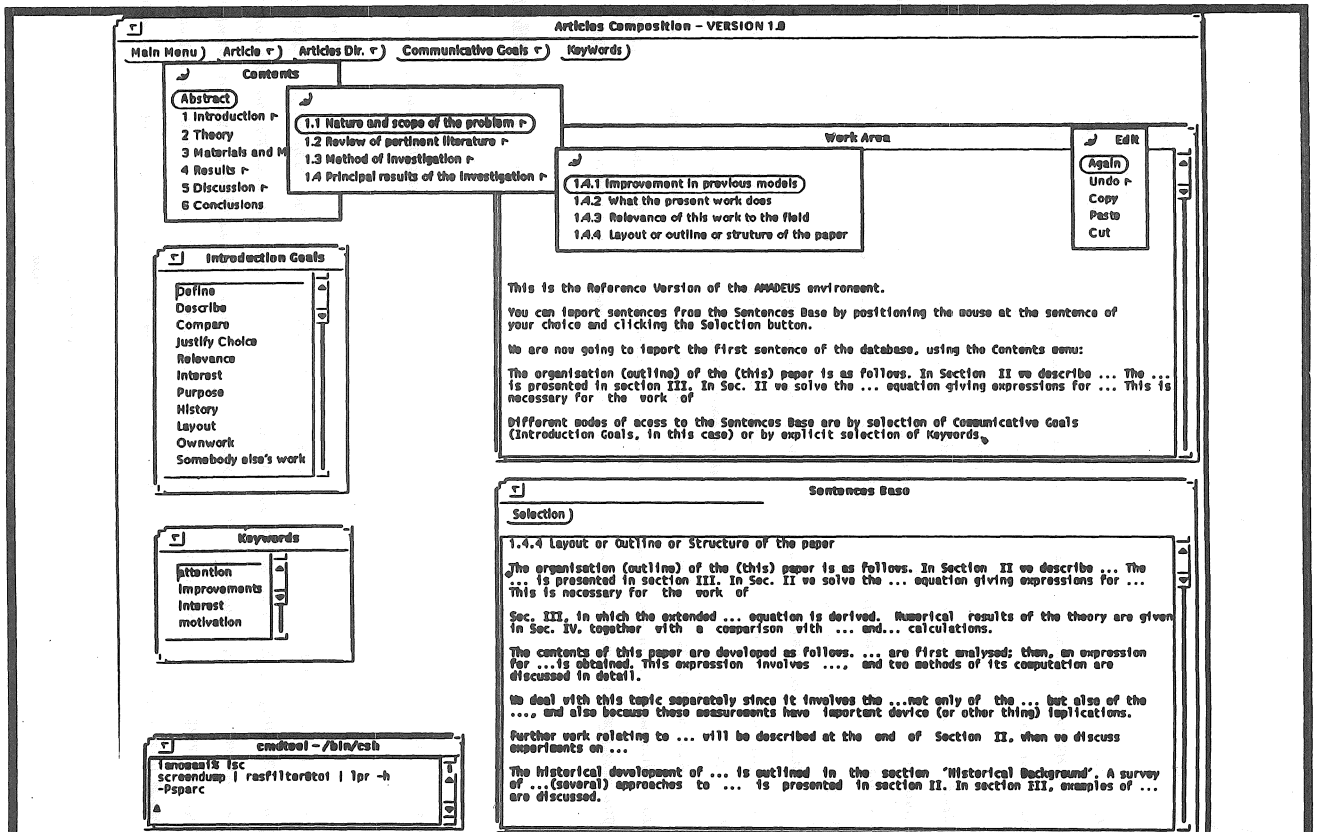


Figure 1: Screendump showing the Reference Version of the AMADEUS software environment.

AN OVERVIEW OF SOFTWARE RISK MANAGEMENT METHODS*

Calvo-Manzano,J.A.; Maté J. L.; San Feliu, T.
Departamento Lenguajes y Sistemas Informáticos e Ingeniería del Software.
Facultad de Informática. Universidad Politécnica de Madrid

Software projects have often suffered in the past from alarming overruns of the budget and schedule, due mainly to the uncertainty of the development schedules and costs estimations. In responding to such question the software industry has gained a great deal of experience in techniques for risk management. More interesting risk management methods are shown. A brief summarize description of the four major methods, its risk definition, phase description and principal characteristics with the relative strengths and weakness of each method is given.

Keywords: Software management, software engineering, risk management, risk assessment, risk analysis.

1. INTRODUCTION

Most organizations are highly dependent on information systems that stand on software and another forms of automatic support. Staying in a competitive environment, which leads to an increasing complexity of the systems to develop, means to arrange a higher degree of automatic support. Yet we must often deal with the following problems: schedule or budget overruns, systems that do not come up to user expectations, money continuing to be spent on completing projects even after it is apparent that the system is defective or is no longer a cost-effective project or systems that work but do not contribute to the organization's goals.

Therefore the risk management area has been expanding over the past years [1]. Why so much concern? One possible answer is: the complexity of software systems is on the increase, and there is a higher interest in a higher quality. Nevertheless it is not a matter of eliminating each and every risk human activity, the goal is not to get risk free technologies, but to determine whether the levels at which we can state are safe enough and the benefits of the activities can overcome possible damages.

When developing a project most analysis of failures show that problems could have been reduced or even solved, if there had been an explicit early concern with to identifying and resolving risk elements. Usually there is excessive optimism during the early phases of the project which lead some clear signs indicating a high risk situation being dismissed.

Modern methods of development make it all too easy for project managers to make high risk agreements they may regret in the future. Sequential life cycle models [2] driven-document like the waterfall model do not provide for the high risk implications arising from the requirements to be established when the contract is made. Experience has shown that successful project managers are good risk managers, although they do not use terms such as risk identification, risk evaluation and planning, however this is precisely what

they are doing. And their projects tend to overcome obstacles and produce good products. This has led to an effort to formalize the applied mechanisms as a "Software Risk Management" discipline.

2. BASIC PRINCIPLES IN RISK MANAGEMENT

We define project risk as the sum of factors that prevent the system from being delivered, meeting the user's requirements or being delivered within a stated time and a previously fixed budget. But this is not the only definition of risk, different authors on risk management methodologies will suggest definitions according to any aspect they want to consider, so risk is defined as "the probability of an undesired event and its impact"[3], or "risk is the traditional manner of expressing uncertainty in the system's life cycle. In a quantitative sense, risk is the probability, at a given point in a system's life cycle, that predicted goals cannot be achieved with the available resources"[4]

Formally, risk is represented by the triplet [5] $<s_i,l_i,x_i>$, where "s" contains the scenarios on what can go wrong, "l" represents the generic likelihood associated to the scenarios and "x" represents the measure of the possible consequences. In order to identify risk areas, we need a classification of risks, whether by their origin, or the point they manifest, or the impact they produce on development. Thereafter, possible couplings and dependences between risk elements must be studied.

Risk management is making informed decisions by consciously assessing what can go wrong and the resulting impact. Risk management is accomplished by a continuous set of activities to identify, confront and resolve risks [6].

The principal phases of each software risk management process are the following:

- Identification. In this phase, potential risk items must be investigated and extracted, as well as their sources, estimations of the associated probability and its effects during a given period. The purpose of these studies is to establish a justification of the costs, in order to select and introduce preventive measures. Risk identification must be done by means of a structural and disciplinary process, in order to identify and analyse systematically the risks.
- Planning. In this phase the information obtained is transformed into decisions and actions (both present and future). Consideration is required to determine: (a) how often monitoring should be performed; (b) by whom; (c) how extensive it should be; and, (d) at what time, and at what level should be the new risk state be set?
- Monitoring. In this phase deviations from planned actions, the behaviour of risks and the appearence of new ones are controlled. Periodic application of risk monitoring is required to judge progress in averting the currently known risks, and to surface any unobservable risks. Results of the monitoring effort should be used to determine the effectiveness of the risk handling techniques.

3. CURRENT METHODS

This paper presents a comparison of the most current important methodologies on risk management today, according to the following scheme for each one: risk definition, phases description and its principal characteristics with the relative strengths and weakness.

- Portfolio Approach to Information Systems.
- Software Risk Abatement.
- Method of Charette.
- Method of Böehm.

3.1. Portfolio Approach to Information Systems

Proposed by McFarlan in 1981, [7] this method identifies risks, provides possible techniques for their resolution, and bases the identification phase on a classification of the organization and its projects.

3.1.1. Risk definition

According to the author, risk means what is left after having applied appropiate managing methods and techniques.

It is distinguished the following risk consequences:

- Failure to obtain all or part of the expected benefits.
- Implementation costs exceed the accepted values.
- Implementation takes longer than expected.
- Technical performance of the systems is lower than expected.
- Incompatiblity of the system with the selected hardware and software.

The author proposes three risk dimensions:

- Project size. Size affects cost, staffing levels, passed time and number of departments the project involves.
- Experience in technology: Due to a higher probability of dealing with unexpected technical problems, the project's risk increases according to the previous experience in hardware, operating system, database management systems and project application language.
- Project structure: In some projects, the very nature of the task is defined at the moment of conceptualization, which provides structural schemes. These are processes that will not be modified during the project's life cycle.

3.1.2. Phase description

The primary steps are:

- Identifying the category of the project among different categories, according to the three dimensions, each one indicating a different degree of risk. It is useful to categorize projects when separating them into their management forms. Risk measurement aims to do this by considering questionnaires developed on the basis of experience in successful or failed projects. The answers to the questions not only make projects

clear, but also suggest alternative ways of project management. If the risk scoring is high, an analysis of the answers may come up with ways of handling risks. At this point, managers must not consider risk as static; moreover, its presence must encourage a better focus better on management.

- Risk control: When, as a result of the previous step, a high scoring is determined, this must be approved by the manager, because this is the way to make sure that managers notice risks and make an assess the consequent benefits. In addition, the aggregate risk of other projects must be determined.
- Project monitoring is done in accordance with planning and control tools. What kind of tools and techniques must be used? These are established depending on the organization and complexity of the project.

3.1.3 Principal characteristics

This method focuses on risk taxonomy, that is the three risk dimensions of a project (size, experience, and structure). Another point of interest is the emphasis on what types of management tools and approaches must be applied to ensure project success.

This approach is focused on personal experiences, coupled with other learning experiences and knowledge of the organization's mission. Risk factors must be at least partially identified from checklists in order to get easyly the information needed for the decision process, because actually the only parameter of the decison process is the risk score. Unlike rest methods, the coverage of all life cycle is not contemplated in this approach.

3.2. Software Risk Abatement. AFSC-800-45

The U.S. Air Force has since been a pioneer in the development and application of risk management techniques. This document summarizes its current approach. The risk management process is an iterative process that requires goals established by the managers, who will regularly evaluate the impediments to achieving these objectives. Risk management must be considered from the point of view of the magnitude of causes and effects of changes.

3.2.1. Risk definition

Risk is the traditional manner of expressing uncertainty in the system's life cycle. In a quantitative sense, risk is the probability at a given point in a system's life cycle that predicted goals cannot be achieved with the available resources[4]. Software risks can be divided into: risk in the performance, support, costs and planning.

3.2.2. Phases description

The software risk management process is divided into:

- Identifying risk components
- Analysing risk factors
- Developing options for dealing with risk and controlling risk factors

- Risk identification. Before starting the process, it is necessary to determine whether software contributes meaningfully to the total risk in the system. Next, risk elements are identified and, at this point, we can obtain information from historic data of similar programs, lessons learned, previous research and precise

studies on similar development programs. Risk items will be identified by analysing these data, as well as its impact on the system.

- Risk components analysis. The analysis of each risk component in the system must be done using techniques from multiple disciplines in order to determine the probability of a risk item and its impact. Impact is divided into four categories: catastrophic, critical, marginal, negligible. Categories are determined by the potential consequences of possible undesired results due to undetected software errors or faults. After the impact and probability of occurrence have been evaluated, the degree of risk can be determined. The seriousness of risks is subdivided into: high, moderate, low or none.
- Management and control of risk. This task manipulates risk factors so that the exposition to risk can be reduced. Risk handling alternatives are divided into four categories: avoidance, control (prevention), assumption and transfer. The selected technique depends on the part of a program where the project falls and on the possible avaible options. After having selected the risk management technique, the results of this technique will be compared with expected ones in order to determine the need of continue with or take possible actions, repeting them when necessary.

3.2.3 Principal characteristics

This method provides a set of guidelines for applying risk management, mainly based on previous USAF experience. Several set of tables for risk identification and risk assessment are given, in order to facilitate the estimation of risk items.

The principal weakness of this approach is that it has been designed in a specifically military environment, however it gives a good set of guidelines for use in a software risk team.

3.3. Method of Charette

3.3.1. Risk definition

Charette defines risk [5] as any event, happening or thing that has a loss associated with it or uncertainty or some choice is involved. The primary sources of risks are lack of recognition or appreciation of risks, lack of the necessary resources, and, lack of productivity of the resources.

3.3.2. Phases description

Charette divides the method for software risk management into two major phases: risk analysis and risk management.

- Risk Analysis. It is the process of identification, estimation and evaluation of risks. Risk analysis is used to identify potential problem areas, quantify risk associated with these problems and generate alternative courses of action that can be taken to resolve risk.

1. Risk identification: Before making risk analysis, it must be considered whether the analysis is necessary. Risk identification is the comprehensive identification of potential risk items using a structured and consistent method, obtaining at the end of this step a categorized list of risks.

2. Risk estimation consists in measuring the probability of potential loss and to risk exposure; during the estimation process the measurement scale will be selected, so that quantitative data about risks can be obtained, trying to reduce the uncertainty of risk estimations.
3. Risk evaluation is a process that makes it possible to foresee answers to risks. The purpose is to foresee the consequences of the different possible alternatives, comparing them with the acceptability of individually predicted consequences to a foreseen decision.

- Risk Management is the planned control of risks and the monitoring of the success of the control mechanisms. Risk management involves making a risk analysis and taking a decision. Three tasks are required to perform risk management.

1. Risk planning. Risk planning is concerned with two principal questions: first, whether the strategy that will be applied for managing risks is feasible and correct; and second, whether the tactics and the means available to implement this strategy fit in with the project's goals.
2. Risk control. Risk control involves the development and evaluation of the feasibility of the implementation of the plan's control mechanisms for the reduction strategies. A control of any contingency that should be in reserve must be established. A risk management plan will be draw up, determining one possible strategy against risks: reduction, protection, transfer, pecuniaries.
3. Risk monitoring. Monitoring comes after the decisions on the reduction strategy and tactics have been already implemented in order to avoid possible risk, that is:
   - Check if the consequences of the decision were the same as foreseen.
   - Identify opportunities for refining of the risk aversion plan.
   - Provide feedback for future decisions concerning the control of new risks or current risks that do not respond to risk aversion, or risks whose nature has changed with time.

3.3.3. Principal characteristics

- This methodology provides severals mechanisms for categorizing risks.
- It also generates a complete documentation: a risk estimation situation document for risk identification and a software risk management plan and software risk aversion plan for risk planning and monitoring phases.

3.4. Method of Böehm

Böehm's software risk management [3] approach of 1989 is based on experience in TRW projects.

3.4.1. Risk definition

Risk is the possibility of loss or injury. This definition can be translated into the concept of exposure to risk, which relates the two basics of risk, probability and possible potential loss. These are factors that will determine possible alternatives to reduce risks, by minimizing probability or impact or both of them.

3.4.2. Phases Description

Risk management practice involves two primary phases and each one with three subsidiary steps.

The first phase, risk assessment, involves risk identification, risk analysis and risk prioritization.

- Risk identification produces lists of the project-specific risk items likely to compromise a project's success. The most common techniques of identification include checklists, examination of decision drivers, comparison with experience and decomposition.
- Risk analysis assesses loss probability and loss magnitude for each identified risk item, and it assesses compound risk in risk-item interaction. The most common techniques are: perfomance models, cost models, network analysis, statistical decision analysis and quality-factor analysis.
- Risk prioritization produces a ranked ordering of the list items identified and analysed risk items. Typical techniques include risk-exposure analysis, risk-reduction leverage analysis, Delphi or group-consensus techniques.

The second phase in risk control involves risk management planning, risk resolution and risk monitoring.

- Risk management planning helps to prepare the strategy for each risk item, including the coordination of the individual risk item plans with each other and with the overall project plan. Most common techniques include checklists of risk-resolution techniques, cost-benefit analysis, and standard risk-management plan outlines, forms and elements.
- Risk resolution produces a situation in which the risk items are eliminated or, at least, reduced. Typical techniques include prototypes, simulations, benchmarks.
- Risk monitoring involves tracking the project's progress toward resolving each risk item and taking corrective action where appropriate. The most common techniques includes milestones tracking, top-ten risk item list, that is reviewed weekly or monthly, followed by reassessment of the risk item, or corrective action.

3.4.3 Principal characteristics

This approach provides a well-defined set of stages, with a well assembled collection of techniques, tools and forms to use in risk management. This framework is flexible to use with several life cycle models, specially designed for the spiral life cycle model [8].

Böehm and Charette's methods focus on similar points of interest:

- Improve qualitative assessment through documentation and reviews.
- Prevention of risk by using highly experienced personnel.

While Böehm's approach works on a unique document (risk management plan) refined by iterative reviews, Charette's approach provides a very different structured documentation, it establishes a document (risk estimation situation) to use in the risk identification phase and two more documents to use in the planning phase (risk management plan and risk aversion plan).

4. SUMMARY

The different methodologies we have presented provide a framework where risk management can be applied in practice. They do not pretend to be the philosopher's stone, nor making success completely sure; nevertheless they are important, because they can help avoiding disasters by focusing on the critical factors of success. There are still points to determine, since risk management needs human judgement on technical and personal factors. In the future, risk management will basically progress in two areas: first, on the new ways of getting software develop process, and, second, on the improvements of risk management techniques. As regards the former, a great effort is being made at the Software Engineering Institute[9] to come up with a process for evaluating software capabilities. Research on the latter is already underway with a view to incorporating new artificial intelligence techniques and fuzzy logic in order to help overcome problems with risk quantification techniques, which will may it possible to improve existing methods. So we must introduce the future changes in improving techniques arising from the experience in applying risk management in more number of projects, and provide a support in the shape of integrated control tools that facilitate an automatic capture of information, filtering and exploring possibles alternatives and solutions to make decisions easier. This work is being carried out in Europe by the EUREKA RISKMAN Project, and in the United States by the System Engineering Risk Management Institute and Software Engineering Institute at Carnegie Mellon.

REFERENCES

[1] M. G. Morgan, "Choosing & Managing Technology-Induced Risk", IEEE Spectrum, Dec 1981

[2] A.M. Davis, E.H. Bersoff, E.R. Comer., "A Strategy for Comparing Alternative Software Development Life Cycle Models", IEEE Transactions on Software Engineering, Oct 1988

[3] B. W. Böehm.Software Risk Management Tutorial. IEEE Computer Society, 1989

[4] U.S. Air Force Systems Command, Software RiskAbatement, AFSC Phamphlet 800-45,1988

[5] R. N. Charette, Software Engineering Risk Analysis and Management", McGraw-Hill 1989.

[6] Department of Defense, Software Technology Strategy, Draft December 1991.

[7] F. W. McFarlan, "Portfolio Approach to Information Systems", Harvard Bussiness Review, Sep-Oct 1981.

[8] J. G. Wolff, "The Management of Risk in System Development: 'Project SP' and the 'New Spiral Model', Software Engineering Journal, May 1989

[9] W.S. Humphrey, "A Method for Assessing the Software Engineering Capability of Contractors", CMU Software Engineering Institute Report CMU/SEI-87-TR23 Carnegie Mellon Univ. Press, 1987

UN SISTEMA DE VISIÓN PARA LA DETECCIÓN DE DEFECTOS

Santiago Candela, Jose A. Muñoz, Francisco Alayon, Carmelo Garcia
Departamento de Informática
Universidad de Las Palmas de Gran Canaria
Las Palmas de G.C, España.

ABSTRACT: This work make a description of techniques and algoritms to define a general strategy to attack the problem of visual automatic detection of defects, based on Systems Theory and Artificial Vision techniques, wich allows for a transportability to defects detection problems on several materials like leather, wood, clothes and plastics.

KEYWORDS: Artificial Visión, Fault Detection, Texture Analysis.

1. INTRODUCCIÓN.

El problema de la detección de defectos aparece en numerosos procesos de fabricación industrial, como un aspecto del control de calidad, por lo que es una faceta fundamental en la automatización de las cadenas de producción. Esto justifica el esfuerzo que últimamente se está realizando en la investigación de técnicas para tal fin.

Cuando nos enfrentamos al problema de la detección de defectos debemos plantearnos, como primer paso en la resolución del problema, el encontrar un marco teórico de trabajo que nos permita plantear y formalizar nuestro problema. Para ello el saber responder a la cuestión *¿Qué se entiende por defecto?* es vital. Aunque en muchos casos, y debido a la alta capacidad de proceso del ser humano, esta respuesta parece sencilla, hoy día desde un punto de vista formal no disponemos de tal respuesta, y por lo tanto, ni mucho menos de un marco teórico general donde abordar de forma analítica este problema.

En este trabajo describimos la estrategia general que hemos diseñado para resolver el problema de la detección automática de defectos en pieles curtidas de animales como cerdo y vaca, y que se utilizan en la industria peletera. Un primer paso consiste en definir que se entiende por defecto, para ello observamos que las imágenes digitalizadas de pieles presentan una cierta uniformidad en su textura, donde la presencia de un defecto se manifiesta como un cambio de textura o una alteración de la uniformidad. Con este planteamiento, el problema de la detección de fallos se transforma en encontrar los cambios de uniformidad o periodicidad en la textura.

2. DESCRIPCIÓN DEL PROBLEMA.

Nuestro objetivo es poder detectar defectos en pieles de animales a partir de las imágenes digitalizadas adquiridas mediante un sistema cámara-computador. Estudiado un amplio muestrario de pieles con defectos, se observa lo siguiente acerca de la apariencia de las zonas defectuosas:

a. Los fallos se manifiestan como cambios estructurales o como cambios tonales.
b. Para un mismo tipo de fallo la intensidad del cambio estructural o tonal con respecto a zonas buenas no siempre es la misma.
c. Incluso en zonas buenas de piel se producen variaciones tonales y estructurales muy suaves.

Dadas estas observaciones, el contestar a la pregunta ¿Cual es el conjunto óptimo de descriptores que nos permiten distinguir y discriminar entre zona buena y zona defectuosa? no tiene fácil respuesta, pues nos encontramos con que tal caracterización es borrosa. Esto va a obligarnos a realizar distintos procesos eurísticos para encontrar una solución al problema.

3. ESTRATEGIA GENERAL.

En este apartado describimos la estrategia general a que hemos llegado, tras la realización de distintas pruebas y que seguimos para resolver el problema. Esta estrategia presenta la ventaja de ser independiente del tipo de material y defecto por lo que es susceptible de ser utilizada en la detección de defectos en otros materiales tales como: madera, telas, plásticos, etc. Nuestra propuesta de esquema está formado por las siguientes etapas:

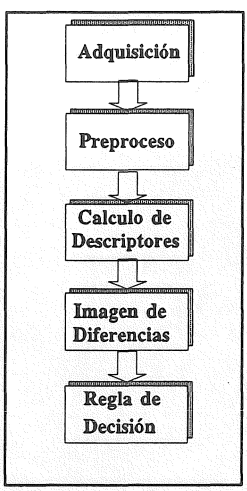


Fig.1 Estrategia General

1. Adquisición de la imagen a analizar.
2. Preproceso de la imagen
3. Obtención de descriptores.
4. Cálculo de la imágenes de diferencias.
5. Toma de decisión.

Fase 1: Adquisición de la imagen a analizar.

Debido a la naturaleza de nuestro problema captamos las imágenes con una camara de estado solido CCD de alta resolución de 1024x1024 pixels, donde cada punto posee un nivel de gris que varía en un rango de 0 a 255. Esta imagen es introducida en el computador mediante una placa de adquisición, almacenamiento y presentación monocroma.

Un aspecto importante relacionado con esta fase son las condiciones luminosas ambientales. Debido a la sutileza visual con que se manifiestan los defectos es primordial que sobre toda la superficie de la piel a adquirir, incida una intensidad luminosa uniforme, pues de lo contrario se pueden inducir defectos no existentes realmente. Otros aspectos importantes que se han puesto de manifiesto en nuestras experiencias es la mejor o peor visualización del defecto en función del tipo de luz empleada y de la dirección de haz de luz incidente.

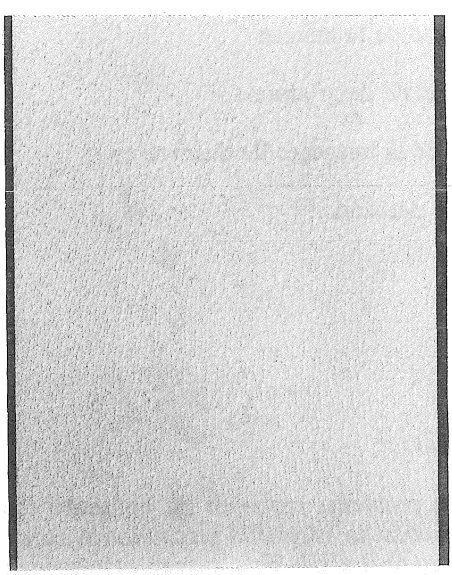
Fase 2: Preproceso de imagen.

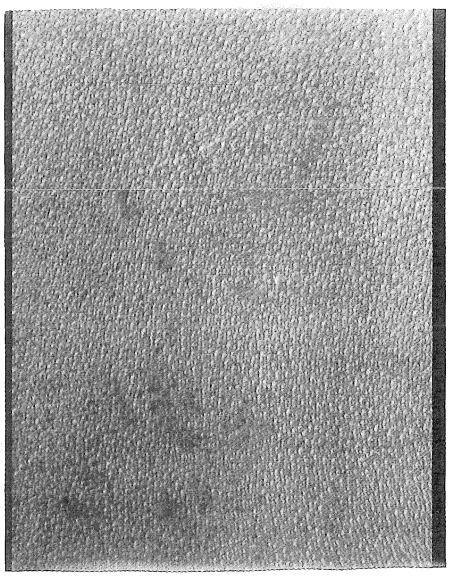
Esta etapa tiene por finalidad, por un lado, lograr la mejora de la imagen a analizar; frente a ruidos o leves variaciones de luz en la fase anterior, y por otro lado, enfatizar los fallos que tiene la piel a analizar; de esta manera se facilita la detección de los defectos.

Entre los preprocesos ensayados uno que ha proporcionado buenos resultados es el conocido por "*proceso facilitador*". Básicamente, se trata de un proceso inspirado en trabajos previos [16] sobre procesamiento retinal de imágenes. Este consiste en evaluar una imagen transformada, R(x,y), como:

$$R(x,y) = C(x,y)*P(x,y) - Q(x,y)$$

Siendo (x,y) las coordenadas de un punto en la imagen; C(x,y) una zona de la imagen original que llamamos centro, alrededor del punto (x,y); P(x,y) otra zona de la imagen original, envolvente de la zona centro C, llamada periferia; y por último, Q(x,y) una función umbral canceladora de ruido.

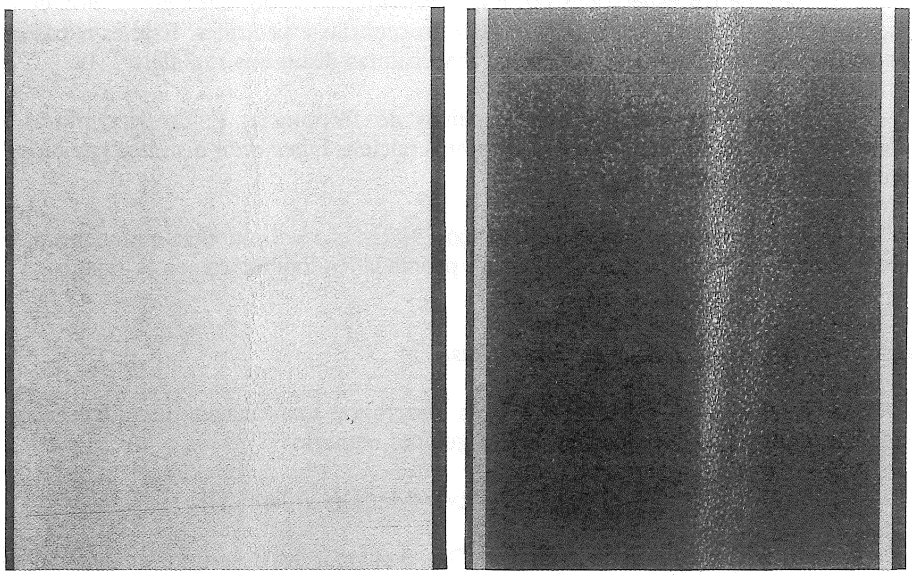




Otro preproceso que proporciona buenos resultados es el "*filtrado paso alto no lineal facilitado por la desviación típica*" ya que pequeños cambios locales en la textura y aumento de la desviación típica van en la misma dirección para indicar variaciones en la textura. Básicamente este filtrado consiste en obtener la imagen transformada, R(x,y), definida como:

$$R(x,y) = Abs(Ev - I(x,y))*Sv$$

Siendo (x,y) las coordenadas de un punto en la imagen; Ev la media tonal de una ventana entorno al punto de imagen; I(x,y) valor tonal en la imagen original; Sv la desviación típica tonal de la ventana entorno al punto de imagen; y Abs es la función valor absoluto (exclusive OR detección de contornos).



Fase 3: Cálculo de descriptores.

En esta fase se calculan los descriptores en ventanas de la imagen, de tamaño fijo, para posteriormente proceder al etiquetado de cada una de ellas como zona buena o defectuosa. Para esto se utiliza un tipo de convolución generalizada [4] sobre la imagen preprocesada, tal que para cada pixel, se definen unos vecinos (el dominio del punto), donde el núcleo de la convolución computa una serie de reglas para obtener un vector de propiedades. Como resultado, obtenemos una serie de imágenes de parámetros, las cuales obviamente muestran el efecto de bordes de toda convolución.

La convolución generalizada puede presentar un coste computacional elevado, tal que para cada caso, es necesario evaluar los requerimientos de computo versus la efectividad del algoritmo. Esta evaluación requiere una serie de cuidadosos test. Diferentes grupos de test se han realizado degradando la "resolución" de la convolución, de forma que se ha ensayado distintos saltos de la posición del pixel central, 2, 4, 16. Los mejores resultados se obtienen para saltos de 1, ya que este proporciona una continuidad

en la imagen, si bien el coste computacional es mayor.

Se ha empleado un amplio abanico de núcleos para la obtención de descriptores de los que destacamos los siguientes:

a. Estadísticos de primer orden [13]: Este conjunto de estadísticos lo constituyen todos aquellos que se obtienen a partir de la distribución de frecuencias de los niveles de gris, este es el caso de la media, varianza, energía, entropía, kurtosis, y apuntamiento.

b. Estadísticos de segundo orden [3] y [18]: Este conjunto de descriptores se obtienen a partir de la matriz de coocurrencia. Este es el caso de inercia, energía, entropía, y homogeniedad local.

c. Parámetros espectrales [7]: Estos descriptores se obtienen en función de los armónicos contenidos en distintas bandas de frecuencias espectrales. Este es el caso de las sumatorias del espectro de potencia para distintas distancias radiales.

d. Respuesta ante núcleos sensitivos de frecuencia: Estos descriptores se obtienen a partir de la respuesta ante distintos núcleos que filtran distintos cambios de frecuencia en los niveles de gris.

e. Respuesta ante núcleos topológicos [9]: Estos descriptores se obtienen a partir de la respuesta ante núcleos que enfatizan propiedades topológicas en la imagen.

Fase 4: Cálculo de la imagen de diferencias.

En esta fase se calcula para cada descriptor una imagen de diferencias o distancias, donde éstas se calculan de la siguiente manera:

$$d(x,y) = (D(x,y) - Dp(x,y))/Min[D(x,y),Dp(x,y)]$$

Siendo (x,y) un punto de la imagen, D(x,y) el valor del descriptor en una ventana central en el entorno del punto, Dp(x,y) el valor del descriptor en un zona periférica ocho ventanas alrededor del punto.

Fase 5: Regla de decisión.

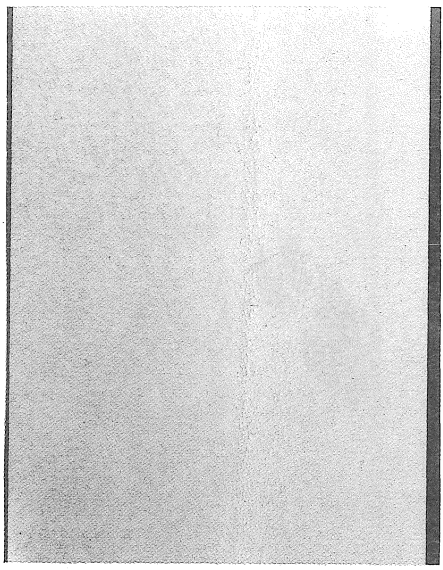
Una vez obtenida la imagen de diferencias para cada parámetro, se procede en esta fase a etiquetar cada ventana de imagen como zona buena o zona defectuosa.

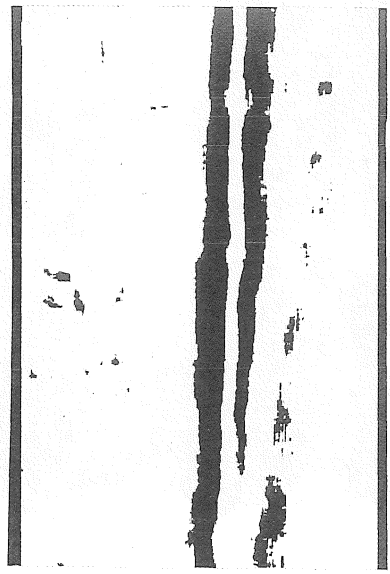
La regla de decisión que se utiliza es:

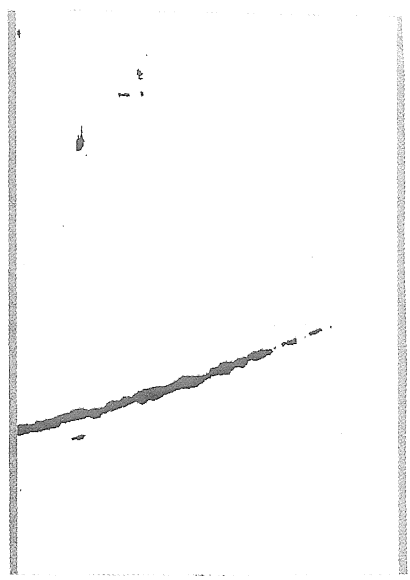
"Si" (d(x,y) > U) "Entonces"
    Zona defectuosa
  "Si no"
    Zona buena
"Fin si"

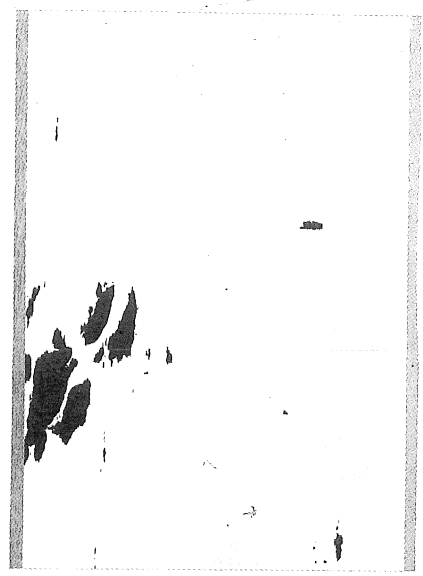
Siendo U un valor umbral característico de cada descriptor.

A continuación se presentan ejemplos donde se ha utilizado como núcleo para la obtención de descriptores los estadisticos de primer orden. La imagen de la izquierda representa la imagen original, y la derecha la imagen resultado con los defectos marcados en negro y la zona buena en blanco.









CONCLUSIONES

La detección automática de defectos en pieles curtidas, utilizando un sistema automático de visión es difícil debido a que el proceso humano de detección es multisensorial (contacto manual de la piel, junto con distintas observaciones visuales) y también por que algunos defectos (los cuales no pueden predecirse con anterioridad) necesitan de unos requerimientos de iluminación especiales para ser vistos por un sistema camara-ordenador. Sin embargo, familias de defectos pueden ser detectados e identificados utilizando algoritmos relativamente fáciles y rápidos.

AGRADECIMIENTOS

Los autores quieren expresar su agradecimiento al Profesor y Maestro Roberto Moreno Díaz por su dirección y aporte de ideas en el trabajo, parte del cual se ha realizado bajo el Proyecto Brite 2195.

BIBLIOGRAFÍA

[1] Amelia C. Fong, Lionel M. Ni, Kwan Y. Wong "Fast Discrimination Between Homogeneous and Texture Regions"

[2] Borghesi, V. Cantoni, M. Diani."An Industrial Application of Texture Analysis".

[3] Calvin C. Gotlieb, Helbert E. Kreyszig."Texture Descriptors Based on Co-Ocurrences Matrices". Computer Vision, Graphics, and Image Processing. Academic Press, Inc. 1990.

[4] Candela, S. Transformaciones de Campo Receptivo Variable en Proceso de Imagenes y Visión Artificial. Tesis Doctoral. Nov 1987.

[5] Conners, R. Identifying and Locating Surface Defects in Wood: Part of an Automated Lumber Processing System. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.Pami-5, No 6, November 1983.

[6] D'Astous and M. E. Jernigan."Texture Discrimination Based On Detailed Measures Of Power Spectrum". IEEE. 1984.

[7] Eklundh, J. O."On The Use of Fourier Features for Texture Discrimination". Computer Graphics and Image Processing. 1979.

[8] Haralick, R. Statistical and Structural Approaches to Texture. Proccedings of the IEEE, vol 67, no 5, May 1979.

[9] Haralik, R. Ridges and Valleys on Digital Images. Computer Vision, Graphics and Image Processing 22, 28-38, 1983.

[10] Kiryati, N.; and D. Maydan. "Calculating Geometric Properties From Fourier Representation". Pattern Recognition. 1989.

[11] Lowitz, G. Images, Princips, Traitements et Applications. Session Extraordinaire de L'ecole Internationale D'informatique AFCET. Abril 1983.

[12] Muñoz Blanco, J. A."Jerarquización de estructuras de nivel bajo y medio para reconocimiento visual. Aplicaciones a texturas y formas. Tesis Doctoral. 1987.

[13] Rao, K., Ahmed, N. Orthogonal Transforms for Digital Signal Processing. IEEE International Conference on Acustics, Speech and Signal Processing P 136-40, 1976.

[14] Richard W. Conners and Charles W. McMillin. "Identifying and Locating Surface Defects in Wood: Part of an Automated Lumber Processing System".IEEE Transactions on Pattern Analysis and Machine Intelligence.1983

[15] Ronsin, D. Barba, S. Raboisson. "Comparision between Co-Ocurrence Matrices, Local Histograms and Curvilinear Integration for Texture". Architectures andAlgorithms for Digital Image Processing. SPIE. 1985.

[16] Santana, O., Candela, S., Moreno-Diaz, R. Computer non-linear and Algorithmic Simulation of Static Retinal Processes. 6th International Congress of Cybernetics and Systems. Paris. 1984.

[17] Steven W. Zucker, Demetri Terzopoulos."Finding Structure in Co-Ocurrence Matrices for Texture Analysis".Image Modeling. Academic Press, Inc. 1981

[18] Unser, M., Coulon, F. Detection of Defects by Texture Monitoring in Automatic Visual Inspection. Proceedings of The 2md International Conference on Robot Vision and Sensory Controls November 1982. Stuttgart, Germany.

[19] Wang, Li.; D. C. He. "A new statistical approach for texture analysis". Photogrammetric Engineering and Remote Sensing, vol. 56, Nº 1, pp 61-66. 1990.

[20] Wang, L.; and D. He. "Texture Classification Using Texture Spectrum". Pattern Recognition". 1990.

[21] Weszka, J. S.; Dyer and A. Rosenfeld."A Comparative Study of Texture Measures for Terrain Classification. IEEE Transaction On Systems, Man, and Cybernetics. 1976.

[22] Young Won Lim, Sang Uk Lee "On The Color Image Segmentation Algorithm Based on the Thresholding and the Fuzzy C-Mean Tecniques.Pattern Recognition.1990

DIARIA: AN INTEGRATED INFERENCE ENGINE

J.P. Caraça-Valente*; A. Gómez-Pérez**; N. Juristo***; J.L. Morant*
* Department of Languages and Systems
** Department of Artificial Intelligence
Universidad Politécnica de Madrid
***Software Engineering Institute
Carnegie Mellon University

Abstract
To date all inference engines have been based on *modus ponens* which is, of course, a deductive inference rule. However, when making inferences, human beings do not only use modus ponens or any other deductive inference rule, they also employ inductive, abductive and deductive inference rules, as required by the circumstances. In this paper, after having studied the different kinds of inference in AI: deductive, inductive, abductive and retroductive, we present a design of an inference engine which incorporates three of these kinds of inference and uses each of them when most appropriate, depending on the data, information and knowledge with which it is supplied at any time. Thus we have a true data-driven inference system.

Keywords
Reasoning models, deduction, induction, abduction, retroduction, hybrid systems

1.- Introduction

The way in which human beings reason remains a mystery. Though we know which reasoning models are used, that is deduction, induction, abduction and retroduction, many questions remain to be answered:

* When is each reasoning model used? Which circumstances lead us to use just one model or a combination of them?
* Why do we use one model and not another? What leads a person to opt for a way of reasoning?
* How do the results obtained by each model overlap? That is, how is it that the conclusions of one inference system may be the starting point for another type of inference and how can we link similar conclusions obtained from different reasoning processes?
* What relationships exist between each reasoning model? Are there hidden relationships that lead to predetermined behaviour with respect to reasoning?
* How can the use of two or more models improve the final results of the reasoning process?
* Are there any combinations of widely used models?

In this paper, we propose a system whose objective it is to go deeper into these questions and thus improve the reasoning methods that can be programmed in a computer, never losing sight of the possibility of answering some of these questions from the human point of view. All four models will be operational in this system (leaving the retroduction model aside for the time being), such that they can interact with each other, share all the knowledge acquired by each one, etc., so as to improve the results on the basis of cooperation.
The paper is composed of this introduction, where the objectives of the system are set out and five other sections. Section 2 analyses previous work on reasoning models. Section 3 describes the design of the system's inference engine and its control mechanisms. Finally, Section 4 gives an example of the system's performance and Section 5 outlines future research.

2.- Reasoning Models

Throughout history, many philosophers and scientists have analysed the nature of intelligence, knowledge and how it is stored in the human brain, trying to discover and sketch a map that reflects the dynamics of thought. Bertrand Russell [1] said that it seemed that Aristotle was the first to explicitly proclaim that man was a rational being. This opinion was based on the fact that some people were able to add up. However, nobody would really consider a machine to be intelligent just because it can perform arithmetic operations. It is its capacity for working with large amounts of data, that is knowledge that can be used to solve problems efficiently or infer new knowledge, that gives it this quality.

An inference is a logic operation by means of which a new fact (proposition, consequence or conclusion) is obtained from other propositions. There are four types of inference, and we will take the following formal schema to examine them:

$$\begin{array}{ll} \forall x\ F(x) \rightarrow G(x) & : R \\ \underline{F(a)} & : F \\ G(a) & : DF \end{array}$$

where R is the rule, F a known fact and DF is a deduced fact. Let's now consider the four types of inference or reasoning models.

2.1 Deduction

Deduction is a process of deriving new sentences or theorems from the sentences of a theory by applying a set of deductive inference rules [2]. Its formal schema is R, E -> DF, that is, a new fact is deduced from the rule and a known fact. Some of the most widely used deductive inference rules are: purely hypothetical syllogisms, modus ponens, modus tollens.

Deduction does not provide any really new knowledge, as all the information obtained was implicitly contained in the premises. Using deduction, however, we can be absolutely sure that the conclusion is true, provided that we are sure that the premises are so.

This is the most suitable reasoning model for using previously acquired knowledge, which means that it is often employed as an inference mechanism in Expert Systems so as to arrive at particular solutions to problems, based on expert judgements. Mycin [3] and Prospector [4] are examples of deductive Expert Systems.

2.2 Induction

Induction is a process of deriving more general hypotheses or rules from individual known facts by applying a set of inductive inference rules. Generally, we have a domain theory and a set of facts to be generalized that meet the conditions of non-redundant facts, unknown conclusion, consistency and generalization.

Its formal schema is F,DF -> R, that is, we could make a generalization on the basis of the pairs of observations (F(a), G(a); F(b), G(b); ...) and obtain the rule "Every F is G".

The most common inductive inference rules according to Gries [5] and Michalski [6] are: standard generalization, restriction of the conjunction, extension of the disjunction, interval closure, specialization tree.

Induction supplies new knowledge that was not contained in the premises, though we have no assurance that the conclusion is true. Nevertheless, provided that we start from a wide and diverse enough set of premises, we can be sure enough about the conclusion to be able to accept it. This really is a creative reasoning model.

Due to its great capacity for obtaining new knowledge, this model has been used in knowledge acquisition tools, usually confined to classification tasks. The most well-known inductive systems are ID3 and successors [7] and AQ [6].

2.3 Abduction

Charniak and McDermott [8] say that abduction is a process of inferring hypotheses that explain given sentences. A hypothesis may in fact be considered as an explanation of another sentence if there is a causal relation between them.
Its formal schema is R, FD -> F, that is, we start from a rule and the consequence of that rule to arrive at the premise of the rule. Abduction supplies new knowledge because we could not deduce the truth of F.
This reasoning model is used in medical diagnostics, for example, where we must abduce -not deduce- the illness causing the symptoms from the latter. There are very few abductive systems, Dendral [9] and RED [10] being the most important examples.

2.4 Retroduction

Retroduction can be seen as second-order induction, which consists in searching for a general abstract law which matches the experimental data. Let´s suppose that we have induced the laws that explain the planets' movement around the sun. If we were to take each one of these laws as a fact and apply induction again, it would be possible to obtain the universal law of gravitation. There are no systems that can be called retroductive. Perhaps Bacon [11] is the nearest there is to retroduction.

2.5 Conclusion

Retroduction is an inference process for creating inventions, scientific laws and general theorems, abduction is a process by which we are able to outline new explanatory hypotheses, induction confirms or refutes such hypotheses and, finally, deduction is an inference mechanism that allows these laws, theories or hypothesis to be used. Induction, abduction and retroduction generate new knowledge about the world but are not formally valid, while deduction only reformulates the knowledge, though it is formally valid.
No knowledge-based system combining all four reasoning models exists as yet. The biggest efforts towards integration have been the predetermined combination of two reasoning models as in the case of learning by analogy and reasoning mechanisms. In this paper, we present a design of an inference engine that works with these four models and provides for any kind of interaction and combinations between them.

3.- System architecture

There are two premises underlying the design of the system, based on the objectives we were pursuing:
- Firstly, we aimed to design as open an architecture as possible. Therefore, we tried not to place any constraints on the behaviour of the system, since we have very little knowledge about reasoning model interaction.
- Secondly, we should make it clear that computational efficiency was not an objective in the early stages of the project. So we were not looking for a system that did things quickly, just that it did them.

3.1 Inference engine design

Taking these two premises as a starting point, we developed the architecture of an inference engine which will support a working memory shared by all the reasoning models. There will only be one knowledge base, where all the facts and hypothesis inferred by the system will be introduced and updated.
In order to achieve maximum flexibility in the system, we decided to divide each inference model into the steps or subgoals that make it up, allowing for the possibility of each step being performed at any moment in combination with any other step of different inference model. This idea will increase the system's possibilities with regard to different reasoning chains.
Figure 1 is an example of how we broke down the first level of the abductive reasoning model [10].

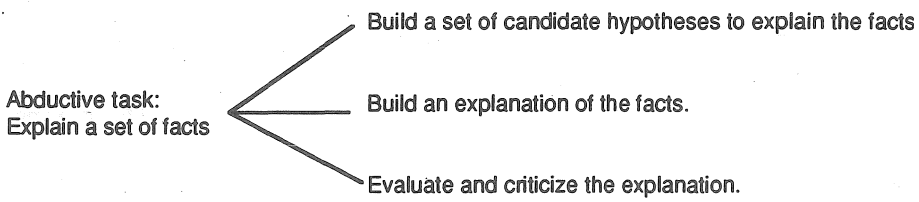


Figure 1

In order to put together an explanation of a set of facts, that is, perform an abductive task, we generally have to take three steps: obtain a set of hypotheses that can explain a general set of facts, select a subset of these hypotheses as the explanation of the facts in question, and finally evaluate and criticize that explanation in order to improve it. In our system, these steps will not necessarily have to be carried out consecutively, any combination with another step of a different reasoning model being feasible.
The other deductive and inductive models have be broken down similarly (we will not take into account the retroductive model in this first version of the system) to arrive at the hierarchy of tasks shown in Figure 2. This Figure shows a two-level decomposition of each reasoning model.

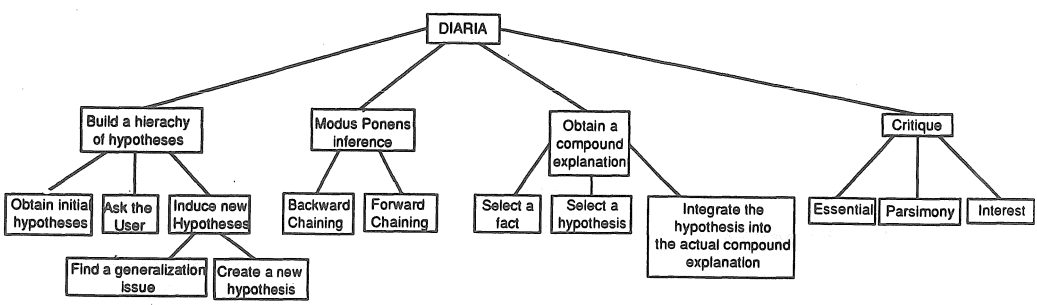


Figure 2

In this Figure, we have grouped tasks with similar objectives under a common branch, irrespective of which reasoning model they belong to.
Let´s examine the purpose of each of these major branches:

* Build a hierarchy of hypotheses module: obtain rules or hypotheses that will later be used to perform abduction or deduction. We have provided for the possibilities of obtaining an initial set from a

previous knowledge base, asking the user for a hypothesis interactively or, finally, inducing new rules from a particular set of facts.

* Modus ponens inference module: We have grouped the two most usual deductive mechanisms, forward and backward chaining, here.
* Obtain compound explanations module: its three components perform the task of finding an explanation of a fact or set of facts.
* Critique module: This groups three types of conclusion evaluation. Essential critique will be used in induction and abduction to determine the need for a characteristic in an induced rule or explanation, parsimony critique which determines whether any explanation is redundant, and interest critique that evaluates the interest of a deductive conclusion or an induced rule by means of characteristics like utility, etc.

3.2 Control mechanisms

The use of control mechanisms is not a contradiction to the objective of achieving a flexible system. These mechanisms will provide for one or other task to be selected, depending on the state of the KB, making it possible to carry out different experiments on reasoning lines.

The control mechanisms must provide a way to evaluate each task´s appropriateness of invocation and then select one task. Each of these tasks will have a sponsor which will indicate at all times the appropriateness of invocation of the task it represents. There is a selector governing this set of sponsors, whose mission it is to select which task has to be performed, taking into account the rates supplied by the sponsors and other knowledge, such as whether a task has ever been carried out, etc.

Each sponsor will have a set of heuristics on the state of the KB, and these will indicate whether it is the moment to perform a given operation. For example, if there are many relations F(a)->G(a), F(b)->G(b), ..., in the KB, it may be the right time to perform an induction to create the rule $\forall x$ (F(x)->G(x)). The set of heuristics needed to define every sponsor is now being developed, this being the main reason for the results shown in next section being provisional.

These sponsor mechanisms mean that the system can be used in three different ways:

* Data-driven strategies: Based on the heuristics of the sponsors, the system will solve the problems it was asked to. This is a true data-driven model which is of great interest as an alternative to traditional instruction-driven models.
* Interactive strategies: Users will be able to define the values for the sponsors and thus analyse variations on a particular reasoning line.
* Predefined strategies: The most useful strategies can be stored in order to examine which reasoning lines are most suited for each problem class. Purely deductive, inductive or abductive reasoning models can also be reproduced.

4.- An Example

The system is still under development, so we cannot offer any definitive results. However, we can give an example of the type of things the system will be able to do.

It goes without saying that human beings don´t use just one reasoning model. Instead, they use the one which gives better results, depending on the circumstances. The example we are going to explain is about an episode in history in the field of scientific discovery, involving a combination of deduction, abduction and induction.

The case starts with the phenomenon of water level in a suction pump repeatedly observed by Galileo. This water level never rose higher

than 10.5 metres. The observation of this strange coincidence led to the formulation of the following inductive rule:

E: water level in a suction pump
A: level higher than 10.5 metres.

$$\frac{E(a) \cap A(a),\ E(b) \cap A(b) \ \ldots}{\forall x\ (E(x) \rightarrow A(x))}$$

In the 18th century, Torricelli, intrigued by the same phenomenon, outlined the hypothesis that the 10.5 ceiling was caused by there being a sea of air that exerted pressure on the water, that is:

S(x) x is a sea of air
R(x) x is around the Earth
P(x) x exerts pressure on the Earth

$$\forall x\ \forall y\ (M(x) \cap R(x) \rightarrow P(x)) \rightarrow (E(y) \rightarrow A(y))$$

This retroductive inference has been simulated in our system through an interactive introduction of this rule through the user facility.
At this point Torricelli carried out another experiment with mercury instead of water. As mercury is 14 times the density of water, the rise in the level of the mercury would be 10.5/14=0.76 m, if the hypothesis were true. The experiment was successful and led Torricelli to perform the next abductive inference:

Hg(y): mercury level in a suction pump
L(y): level to 0.76 m.

$$\frac{\forall x\ \forall y\ (M(x) \cap R(x) \rightarrow P(x)) \rightarrow (Hg(y) \rightarrow L(y)) \qquad Hg(a) \rightarrow L(a)}{M(b) \cap R(b) \rightarrow P(b)}$$

The probability of Torricelli´s theory being true has increased, and from here we can deduce, making only minor changes, the level of any liquid element in a suction pump, if we accept that the abduction is valid.

5.- Future work

The main priority of our work in the short term is to implement the full system and finish defining the heuristics for all the sponsors, as well as carry out a large number of experiments in several predefined domains so as to obtain conclusions on the objectives of the systems.
In the future, we plan to enlarge the system to encompass retroductive inference - we are now doing some work in this respect -, and another important idea is to design a learning module that would be able to learn decisions on improving reasoning strategies in particular domains by tracing the operation of the system.

Acknowledgments

The authors would like to thank Eloy Rodero and Ramiro Lago for their work in putting together the example proposed to the system.

[1] Russell, B; "La Filosofía científica"; Fondo de Cultura Económica; Mexico; 1953
[2] A. Thayse et al; "Approche logique de L´Intelligence Artificielle"; Dunod,1991
[3] Shortliffe, E.H; "Computer Based Medical Consultations Mycin"; New York American Elsevier; 1976.
[4] R.O. Duda, R.E. Hart, K. Konolige and R. Rebon; "A computer based Consultant for mineral exploration"; Technical Report; SRI International; Sept 1979.
[5] D. Gries; "The Science of Programming"; Springer-Verlag; Beril; 1981
[6] R. Michalski; "A Theory and Methodology of inductive learning" in Machine Learning: An Artificial Intelligence Approach; Vol 1; 1984
[7] Quinlan, J.R; "Learning Efficient Classification Procedures and their Application to Chess End Games" in Machine Learning: An Artificial Intelligence Approach; 1984
[8] E. Charniak, D. Mc Dermott; "Introduction to Artificial Intelligence"; Addison-Wesley; 1985
[9] B.G. Buchanan, G.L. Sutherland and E.A. Feigenbaum; "Heuristic Dendral: a program for Generating explanatory hypothesis in Organic Chemistry"; Edinburgh UK,Edinburgh Univ. Press; 1969
[10]Punch III and al; "Peirce a Tool for Experimentation with Abduction"; IEEE Expert; October 199
[11]Pilangley, G.L. Bradshaw, H.A. Simon; "Rediscovering Chemistry with the Bacon System"; Machine Learning; Michalski et al (eds); 1983

Especificación y modelaje de diseños en CAD

Rodrigo Cardoso
Olga Lucía Cardoso
José Tiberio Hernández
Oscar Ramos

e-mail: rcardoso@andescol.bitnet
jhernand@andescol.bitnet
dfac@andescol.bitnet

Grupo DFAC
Departamento de Ingeniería de Sistemas y Computación
Universidad de los Andes
Bogotá, D.E., Colombia

Resumen

Se presenta un marco conceptual dentro del que se define formalmente la dinámica del proceso de diseño. En él puede abordarse con claridad la implantación de ambientes de diseño apoyados computacionalmente (CADE). Los ambientes que resultan son interactivos, adaptivos, de diseño descendente, y su implantación parece natural en el paradigma de programación orientada por objetos.

Palabras claves

CAD, CAM, diseño, bosquejo, rasgo, diseño descendente, diseño adaptivo, reutilización, programación orientada por objetos.

1 INTRODUCCION

Los sistemas actuales de CAD/CAM son fundamentalmente herramientas de ayuda en la documentación y validación de diseños terminados. El proceso mismo de diseño carece de herramientas de apoyo y debe adelantarse sin mayores ayudas computacionales. Lo usual es introducir el diseño al sistema, validar algunos aspectos mediante herramientas de análisis y aprovechar la informática para su eventual reutilización ([ElD90],[Tom87]).

Siendo 'objetos terminados', los diseños se centran en una representación geométrica enriquecida con una descripción de atributos no gráficos. Las herramientas CAD pretenden una genericidad capaz de ser útil en todos los dominios imaginables.

El proceso real de diseño es una aproximación gradual a la especificación completa de un objeto que satisfaga requerimientos impuestos desde un principio o aun dentro del proceso en sí. El objeto terminado es el elemento final de una secuencia de bosquejos, en la que los detalles definitivos se establecen gradualmente. Durante el proceso se elaboran y sopesan alternativas de bosquejos, y se toman decisiones que determinen el rumbo definitivo del diseño. Por otra parte, el diseño es una labor multidisciplinaria en la que cooperan personas que estudian diferentes aspectos, que deben integrarse armónicamente tanto en los bosquejos como en el producto final [You92].

Para solucionar las deficiencias mencionadas de los sistemas actuales, de suerte que verdaderamente se apoye el proceso de diseño, proponemos un marco conceptual en el que se puedan definir sistemas CAD que llenen los vacíos anotados. Más precisamente, recomendamos el uso de sistemas interactivos de apoyo al diseño adaptivo descendente, que puedan sintonizarse a dominios específicos.

La sección 2 establece conceptos básicos que permiten entender con precisión la labor de diseño, sus partes y su desarrollo. En la sección 3 se comentan, de manera general, aspectos que deben ser tenidos en cuenta al enfrentar la implantación de ambientes de diseño que se apoyen en las ideas de 2. Se señala el paradigma orientado por objetos (OO) como la plataforma natural para llevar a cabo dicha implantación [Tom89]. En la sección 4 se resumen conclusiones relevantes. Deliberadamente se ha excluido -en aras de la brevedad- la ilustración, mediante ejemplos, de muchos de los conceptos tratados. Los autores atenderán gustosamente cualquier inquietud o aclaración que se estime necesaria.

2 CONCEPTOS BASICOS

Llamaremos *producto* a una descripción completa del objeto que se pretende diseñar. Todos los atributos y funciones relevantes y sus interrelaciones deben ser precisamente establecidos. Adicionalmente, se desea que los productos sean "fabricables", i.e., realizables en el contexto de un ambiente de fabricación entendido.

Un *bosquejo* es una descripción incompleta de un producto. Entre bosquejos se puede definir una relación de orden `.<.`, con el significado de "menos definido que", i.e., $b_1 < b_2$ establece que el bosquejo b_1 es menos detallado que el bosquejo b_2. Menos detallado puede querer decir, que en b_2 hay algún atributo que en b_1 no está definido, o lo está parcialmente.

Un bosquejo `b` *representa* todos los productos que puedan ser igual o más definidos que él. Este conjunto se llamará el *dominio* de `b`, y se denotará `dom(b)`. El conjunto de los bosquejos se denominará `Bosquejos`.

El *(proceso de) diseño* se entenderá como un grafo finito dirigido acíclico en el que los nodos son bosquejos y los arcos reflejan la relación de orden `.≤.`. Un producto es, entonces, un nodo terminal de un diseño, que sea fabricable. El conjunto de los diseños se denotará `Diseños`.

Antes de diseñar (i.e. construir diseños), se entiende que se ha establecido un conjunto de *requerimientos* que un producto deberá satisfacer. En el mismo proceso del diseño puede ser necesario refinar o detallar en algún sentido lo requerido.

2.1 Derivación de diseños

El trabajo del diseñador consiste en construir diseños a partir de requerimientos. Comienza con un diseño *vacío* (i.e. un grafo vacío), al que se van agregando bosquejos y arcos entre ellos. La decisión de agregar un nodo depende de los requerimientos iniciales así como de las restricciones que puedan resultar de otras decisiones que se hayan debido tomar anteriomente.

Los nodos fuente en un diseño -a los que no llegan arcos- son bosquejos *primitivos*. Son conocidos para el diseñador, en el sentido de que no se discute su procedencia ni su construcción. Un bosquejo primitivo puede ser un tipo básico usualmente disponible en un lenguaje de programación, v.gr. números, cadenas de caracteres, listas, pilas, etc. En un dominio de diseño específico resulta de gran utilidad el disponer de bosquejos primitivos más elaborados que los ya mencionados. Por ejemplo, en el diseño de sistemas hidráulicos, resultaría útil contar con bosquejos primitivos como "bomba", e incluso "bomba rotacional" (y dentro de ésta: "bomba centrífuga", "bomba axial", etc.).

Entre los bosquejos primitivos se considera el bosquejo *vacío* . Con él se representan todos los productos posiblemente diseñables.

A partir de un diseño dado `D`, se puede construir un diseño *derivado* `D'`, agregando un nuevo nodo a `D`. Además de la posibilidad de agregar un bosquejo primitivo, es factible añadir un bosquejo más definido (i.e. con más detalles, atributos o funciones) que alguno de los de `D`. Así, el diseño derivado `D'` puede, también, considerarse como "más definido" que `D`. En este caso, se dice que el nuevo bosquejo agregado es *complejo* , y su definición exacta depende de la de sus predecesores en el grafo.

En el caso general, sean $b_1,\ldots,b_r$ bosquejos del diseño dado D, y b el bosquejo que se desea agregar, de manera que $b_i<b$, para $i=1,\ldots r$. Se consideran tres posibilidades para llevar a cabo este proceso de *derivación* :

- *Enriquecimiento* : En este caso, $r=1$. El nodo b representa un bosquejo en el que se puede calcular una función f que no era considerada en el bosquejo b_1. Las demás funciones que operaban sobre b_1 son heredadas por b.
  Para este caso, se dirá que $b = enr(b_1,f)$, y además, $dom(b) = dom(b_1)$.
- *Composición*: : En este caso, $r>1$. El nodo b representa un bosquejo que junta todos los atributos considerados en los bosquejos $b_1,\ldots,b_r$, que son llamados sus *componentes*. Se dirá que un *rasgo* (inglés: *feature*) de un bosquejo es una componente, o bien, una componente de un rasgo. Las funciones que operaban sobre cada b_i deben extenderse de manera que ahora operen sobre b.
  Para este caso, se dirá que $b = comp(b_1,\ldots,b_r)$, y además $dom(b) = dom(b_1) \times \ldots \times dom(b_r)$.
- *Restricción* :: En este caso, $r=1$. El nodo b representa un bosquejo en el que se exige que uno o más atributos ya considerados en el bosquejo b_1 satisfagan una relación p. Las funciones que operaban sobre b_1 son heredadas por b.
  Para este caso, se dirá que $b = rest(b_1,p)$, y además $dom(b) = \{u \in dom(b_1) \mid p(u)\}$.

Hay una forma especial de restricción, denominada *conexión*. En ésta, el bosquejo b_1 es el resultado de una composición, v.gr. $b_1 = comp(a_1,\ldots,a_n)$. La relación de restricción p se refiere exactamente a una de las componentes, digamos a_k. Además, se quiere que $p(a_k)$ sea satisfecha por exactamente un elemento. Este único elemento debe ser un bosquejo primitivo fabricable, y se dice de él que es una *componente conectada*. Como notación, se dirá que $b = conx(b_1,k,p)$.

La notación introducida permite referirse a los bosquejos por su forma de construcción en términos de otros bosquejos menos definidos. En particular, los productos tienen como referencia una cadena de derivación que refleja la forma en que pueden llegar a realizarse.

2.2 Vistas de un diseño

En problemas complejos, en los que es necesaria la intervención de un grupo de diseñadores especialistas en diferentes áreas, o simplemente en los que un diseñador único desee dividir su trabajo según ciertos criterios, surge el concepto de *vista* de un bosquejo, y, por extensión, del diseño.

Una vista es una función $v:Bosquejos \to Bosquejos$ que "extracta" algunos aspectos "interesantes" de los bosquejos. Por ejemplo [Män90], si se tiene un bosquejo de una lámpara, una vista interesante puede ser su estructura, otra, su sistema eléctrico, etc. Nuestra definición de *vista* es aun más exigente: se desea poder explicar con precisión el significado de "extractar" aspectos interesantes.

Si v es una vista, el hecho de que `v(a)=a'` significa que `a'` tiene algunos de los rasgos de `a`. Además, las funciones de `a`, cuyos dominios se formen exclusivamente con rasgos extractados, son heredadas por `a'`. Obsérvese que, el bosquejo `a'` resulta estructuralmente similar al bosquejo `a`; de hecho, los dos bosquejos son iguales, salvo los aspectos olvidados en la extracción. La noción de vista se puede extender al conjunto de diseños, i.e. como función `v:Diseños→ Diseños`. Si `v(d)=d', d'` es un diseño análogo a `d`, en el que se han extractado los aspectos relevantes de cada uno de los bosquejos que conforman a `d`[1].

3 AMBIENTES DE DISEÑO DE PROPOSITO ESPECIFICO

El marco conceptual expuesto en la sección anterior permite plantear características generales, deseables en ambientes de diseño, asistidos por computador. Llamaremos CADE a un tal ambiente (inglés: *Computer Aided Design Environment*). Un CADE haría las veces de un administrador de diseños, permitiendo su evolución en la forma ya anotada. Tal funcionalidad incluiría, entre otras labores: validación de bosquejos, derivación de diseños, administración e integración de vistas, propagación de restricciones, reutilización de bosquejos, etc.

Es apenas natural esperar que un CADE sea *interactivo*. La actividad de diseñar es esencialmente una labor de 'edición' de bosquejos y diseños, que debe ser apoyada por el ambiente de manera flexible y amable para el usuario. Por otra parte, un CADE debe ser *adaptivo*, i.e. debe tener la posibilidad de apoyarse en bosquejos o diseños existentes para adecuarlos a otras situaciones, dependientes de los requerimientos en cuestión. En otras palabras, debe *reutilizar* soluciones ya conocidas.

La definición de diseño, como grafo dirigido de bosquejos, cuyos arcos significan "igual o más definido que", realiza una concepción inherentemente *descendente* (inglés: *top-down*) de la labor de diseño. La derivación de un diseño se apoya en la definición de otro menos definido. Este modo de trabajo no excluye la posibilidad de considerar, dentro de un diseño, alternativas que permitan explorar distintas maneras de satisfacer los requerimientos dados. En particular, es posible considerar, a partir de un bosquejo no terminal, formas de derivación diferentes de las ya establecidas. Es decir, las acciones de *reintento* (inglés: *backtracking*) pueden adelantarse naturalmente.

Un CADE debería administrar *versiones* de bosquejos, de manera que el diseñador pudiera comparar cómodamente alternativas. Esta característica es particularmente deseable cuando pueden concebirse varios productos-versiones, y debe elegirse uno de ellos para ser fabricado [Mar90].

---

[1] Algebraicamente se puede definir, en forma natural, un homomorfismo `v':Bosquejo→Bosquejo`, tal que, si `v(a)=a'`, entonces `v'(a')=a`. La extensión de vistas a diseños hace factible la extensión de `v'` a un homomorfismo de grafos.

Los bosquejos primitivos que sirven de comienzo en la derivación de diseños pueden ser de propósito general (v.gr., el bosquejo vacío, los tipos de un lenguaje de programación). No obstante, si sólo éstos fueran permitidos, la reutilización de soluciones ya conocidas distaría mucho de ser verdaderamente automatizable, y muchos diseños deberían "reinventar la rueda". Por tanto, es deseable que un CADE manipule bases de bosquejos primitivos, en donde resida el conocimiento que refleje la experiencia de los diseñadores.

Más aun, un CADE de propósito general, i.e., en el que pudiera diseñarse cualquier producto, parecería demasiado ambicioso para ser considerado como una buena herramienta. Un CADE de *propósito específico* serviría en la elaboración de diseños para dominios más pequeños y precisos. Así, además de disponer de bases de bosquejos primitivos adecuados a las posibles necesidades, podría ser factible operar los bosquejos/diseños de maneras quizás especializadas, distintas de las consideradas en 2.

3.1 Hacia una implantación de un CADE

En primera instancia, habría que pronunciarse sobre la plataforma tecnológica en la que podría implantarse un CADE. Las definiciones y consideraciones ya mencionadas parecen apuntar en forma natural hacia el tratamiento del problema de diseñar con herramientas *orientadas por objeto* [Tom89]. En este sentido, se explicará en forma breve cómo los conceptos definidos pueden refrasearse en términos como clases, objetos, métodos, herencia, etc., de forma que su manipulación también resulte natural en este contexto.

Un bosquejo puede representarse por una clase. La relación `.>.` (más definido que) corresponde a la relación de "ser subclase de"[2]. El dominio de un bosquejo es el conjunto de todas sus posibles instancias, como clase. La clase universal representa el bosquejo vacío, y un producto es una clase que sólo tiene una instancia posible.

Los atributos y funciones de bosquejos son atributos y métodos de las clases correspondientes. En un producto están "totalmente implantados", i.e., corresponden a objetos predefinidos y operaciones ejecutables en la plataforma computacional de que se disponga. En un bosquejo que no sea producto se tendrán "implantaciones parciales", i.e., los atributos o métodos están definidos de manera paramétrica[3], de suerte que la instanciación de los parámetros con productos correspondientes resulte en una instancia que sea, a su vez, un producto.

---

[2] Se está entendiendo que la clase `b'` es subclase de la clase `b` cuando la definición de `b'` incluye todos los atributos y métodos definidos para `b`.

[3] En la terminología de objetos, se habla de *genericidad*, y su implantación se logra a través de las llamadas *clases abstractas*.

Los bosquejos primitivos corresponden a clases predefinidas o disponibles al comenzar la tarea de diseñar. Las operaciones de derivación consisten en construir una clase (bosquejo) b a partir de la definición ya disponible de otras clases:

- *Enriquecimiento* : Si `b = enr(`b_1`,f)`, entonces la clase b es una subclase b_1, a la que se aumenta la definición del método f. Este puede ser genérico.
- *Composición*: Si `b = comp(`$b_1,\ldots,b_r$`)`, la clase b hereda los atributos y métodos de las clases $b_1,\ldots,b_r$.
- *Restricción*:: Si `b = rest(`b_1`,p)`, la clase b hereda los atributos y métodos de la clase b_1, pero sólo son instancias de b aquéllas de b_1 que satisfagan el predicado de restricción p. En la práctica, el predicado p debe ser expresable y calculable en el ambiente de programación disponible.

El control de un CADE así implantado debería administrar los requerimientos que deben satisfacerse, tanto los inicialmente planteados, como los que surjan a lo largo del proceso. Por ejemplo, es importante que cualquier bosquejo que sea definible sea en algún sentido realizable, i.e. debe poder exhibirse un producto que sea instancia de él. Una manera de garantizar la *consistencia* del diseño podría considerar exigir del usuario la definición de un ejemplo que satisficiera las condiciones que se imponen sobre un bosquejo, en cada derivación por restricción.

El manejo de *alternativas* aparece al permitir que un bosquejo tenga más de un sucesor posible. Las variantes que así surgen son, en el supuesto de que los requerimientos sean cabalmente atendidos, soluciones diferentes para el problema planteado. La selección de una mejor puede depender de criterios adicionales, considerables quizás sólo al final del proceso.

3.2 Manipulación de vistas

Extendida a diseños, como se sugiere en 2.2, la noción de vista sirve para entender, separadamente, diferentes aspectos del diseño de un producto. Esta cualidad parece plausible, en especial cuando el diseño ya se entiende como un todo, acabado. Sin embargo, los beneficios que deberían derivarse de la posibilidad de definir vistas tendrían que ser capitalizables *durante el proceso* de diseño. Así, un CADE que maneje vistas deberá coordinar, dinámicamente, la evolución paralela de los procesos parciales de diseño que representan diferentes vistas de un solo diseño.

Hay problemas de coordinación que pueden presentarse, análogos a los que pueden existir en la manipulación de información que resulta significativa para varios usuarios, que la utilizan de manera simultánea[4]. De hecho, dos vistas pueden compartir rasgos de un bosquejo, y si una de ellas impone condiciones sobre estos rasgos (v.gr. los

---

[4] Este es el problema de la coherencia de la información de una base de datos con usuarios que pueden efectuar transacciones de modificación, en principio simultáneamente.

define, los compara, etc.), la otra debería conocer tales restricciones y verificar su factibilidad. Naturalmente, debería asegurarse que las dos vistas no intentan modificar o restringir, al mismo tiempo, los rasgos comunes.

Así, si se considera un *diseño global* , definible, en algún sentido, como la unión de varias vistas, un CADE debe tener en cuenta que

- el diseño global debe reflejar los cambios en los diseños vistas,
- los diseños vistas deben conocer las condiciones sobre rasgos que los afecten, que sean causados por otras vistas,
- la consideración de alternativas en diseños vistas conlleva la consideración de alternativas en el diseño global y en otros diseños vistas,
- una vista puede vetar una alternativa -quizás sugerida por otra vista- si no se puede encontrar un producto que realice, en el contexto de la primera vista, el bosquejo resultado de considerar las nuevas restricciones.

La implantación de mecanismos de control y coordinación que brinden los servicios de definición de alternativas y vistas tiene como principal dificultad la *propagación de restricciones* entre vistas, así como la explosión combinatoria del número de bosquejos-alternativas que resultan de considerar variantes en varios bosquejos-vista que compartan rasgos de definición. Para intentar remediar o aliviar estas dificultades, se debería, en lo posible:

- definir vistas independientes, i.e. que no compartan rasgos,
- limitar la definición de alternativas, dentro de diseños-vistas. O bien, considerar alternativas "internamente", dentro de cada vista, y elegir el menor número posible de ellas, antes de comunicarlas a otras vistas con rasgos comunes, a través de los mecanismos de concertación que en este caso sean necesarios.

3.3 Sobres de presentación

La reutilización de bosquejos se constituye en la forma, por excelencia, de ahorrar esfuerzos y costos dentro del proceso de diseño. Un CADE debe administrar almacenes de bosquejos, que puedan ser consultados y reutilizados por los usuarios, de manera cómoda [Ram91].

En muchas ocasiones, el diseñador no necesita enterarse, en todo posible detalle, de las características de los productos que un bosquejo describe. En realidad, interesan mucho los detalles de interrelación con el exterior (v.gr. costo, número de conectores, dimensiones, disipación de calor, etc.), y poco o nada, los de la constitución interna (materiales, componentes internos, etc.).

Para un bosquejo, un *sobre de presentación* es una vista que extrae rasgos relevantes para la reutilización del bosquejo en diseños que no pretendan explicar el funcionamiento interno de los correspondientes productos. Por ejemplo, los productos que puedan ser conectados, y que por tanto no deben explicarse ni entenderse en su funcionamiento interior, deben tener sobres de presentación que mencionen únicamente sus atributos relevantes "para el exterior".

Un CADE debería contar con facilidades de consulta, manipulación y almacenamiento de bosquejos, y proveer estos servicios a través de un manejo eficaz de un *catálogo de sobres de presentación*.

3.4 Ambientes de propósito específico

Aunque se recomienda que un CADE se entienda especialmente útil para elaborar diseños dentro de un dominio específico, hay aspectos genéricos a cualquier CADE concebible, que pueden pensarse como constituyentes de un CADE *núcleo*. Este debe contar con bosquejos básicos análogos a los tipos de un lenguaje de programación (números, cadenas de caracteres, etc.). Todos los mecanismos de derivación de diseños, así como los servicios de edición de bosquejos (crear, guardar, etc.), deben ser disponibles.

En la concepción de un CADE de propósito específico debe adelantarse una labor de profesionalización de los servicios al área particular en que se desea diseñar. Así, un CADE orientado a resolver los problemas de un dominio específico resultaría de completar el CADE núcleo, logrando su especialización a través de

- establecer jerarquías de clasificación de bosquejos por rasgos relevantes al dominio,
- extender los servicios del núcleo, para manipulación de bosquejos especializada al dominio,
- considerar bosquejos primitivos especiales al dominio,
- definir y manejar vistas propias del dominio.

4 CONCLUSIONES

Se presentó un marco de ideas donde pudieron definirse conceptos relativos a la actividad del diseño -producto, bosquejo, diseño, rasgo, vista, versión-. Una óptica formal de tales nociones da sustento a la construcción de verdaderos ambientes de diseño apoyados por herramientas computacionales (CADE). Es importante resaltar que la labor de diseñar se ha formalizado pretendiendo modelar formas usuales de trabajo, de modo que se atienda, además del producto, al proceso de diseño como tal.

El enfoque utilizado pretende integrar aspectos expuestos en los modelos orientados por rasgos (inglés: *feature oriented*) de [Sre88], de ambientes CAD inteligentes e interactivos de [Vee89], de diseño descendente [Män90] y de ingeniería concurrente [You92]. Algunas características son motivadas por la praxis y otras por ideas propuestas, principalmente en [Yos83].

El paradigma OO se ha sugerido como plataforma tecnológica para el desarrollo de CADEs. En él se han refraseado los conceptos generales, que incluso esbozan ideas sobre la definición de un CADE núcleo, que sirva de piedra angular para la construcción de otros, de propósito particular. Por lo demás, la realización de CADEs debe dar luces que orienten la construcción de otros, y esto debe ser particularmente cierto al intentar teorizar sobre la mejor forma de administrar la reutilización de bosquejos en dominios específicos.

El tratamiento formal y el manejo unívoco de los conceptos considerados señalan pautas claras para adelantar dentro de ellas la labor de implantación de CADEs, que actualmente se encuentran en curso en el laboratorio del grupo DFAC.

BIBLIOGRAFIA

[Car91] Cardoso, R., *Verificación y desarrollo de programas* , Ediciones Uniandes, Bogotá, 1991.

[dlR91] de la Rosa, F., Hernández, J.T., CAD de propósito específico: una propuesta por rasgos, Memo de Investigación, CIFI, Universidad de los Andes, Bogotá, agosto 1991.

[ElD89] El Dashan, K., Barthes, J.P., Implementing Constraint Propagation in Mechanical CAD Systems, en *Intelligent CAD Systems II*, Akman, V., ten Hagen, P.J.W., Veerkamp, P.J. (eds.), Springer Verlag, 1989.

[Her92] Hernández, J.T., Rodríguez, M., Gutiérrez, E., Ferreira, A.C., Zendron, P.C.D., IAGE: A feature oriented CAPP System, 6th Conference on CAD/CAM, Robotics and Factories of the Future, Bera, H., Gill, R. (eds.), Southbank Press, 1992.

[Män90] Mäntylä, M., A modeling system for top-down design of assembled products, IBM Journal of Research and Development, Vol. 34, No. 5, Septiembre 1990.

[Mar90] Marín, R., Diseño conceptual adaptivo en sistemas mecánicos, Tesis M.Sc., Universidad de los Andes, Bogotá, abril 1990.

[Ram91] Ramos, O., Hernández, J.T., Ambientes de diseño y modelaje orientado por rasgos, Memo de Investigación, CIFI, Universidad de los Andes, Bogotá, agosto 1991.

[Sre88] Sreevalsan, S.P., Rogers, M.T., Billo, R., Mathew, A., *Current status of feature technology, 2nd. edition* , CAM-I Report R-88-GM-04, Arlington, 1988.

[Suh90] Suh, N.P., *The principles of Design*, Oxford University Press, New York, 1990.

[Tom87] Tomiyama, T., ten Hagen P.J.W., The Concept of Intelligent Integrated Interactive CAD Systems, CWI Report, Centre for Mathematics and Computer Science, Amsterdam, abril 1987.

[Tom89] Tomiyama, T., Object Oriented Programming Paradigm for Intelligent CAD Systems, en *Intelligent CAD Systememes II*, Akman, V., ten Hagen, P.J.W., Veerkamp, P.J. (eds.), Springer Verlag, 1989.

[Vee89] Veerkamp, P., Akman, V., Bernus, P., ten Hagen, P., IDDL: A Language for Intelligent Interactive CAD Systems, en *Intelligent CAD Systems II*, Akman, V., ten Hagen, P.J.W., Veerkamp, P.J. (eds.), Springer Verlag, 1989.

[Yos83] Yoshikawa, H., CAD framework guided by general description theory, en *Proceedings of the IFIP WG 5.2 Working Conference on CAD systems framework*, North-Holland, Roros, 1983.

[You92] Young, R.E., O'Grady, P., Greef, A., A Concurrent Engineering System: The Use of Constraint Networks, 6th Conference on CAD/CAM, Robotics and Factories of the Future, Bera, H., Gill, R. (eds.), Southbank Press, 1992.

Cover Letter

Weightless Neuron Models

Edson Costa de Barros Carvalho Filho
Universidade Federal de Pernambuco
Departamento de Informática
Recife — PE — Brazil
CEP 50.739
e-mail:edson@di.ufpe.br

Topic: Neural Networks

Paper accepted to the XVIII Latinamerican Informatics Conference to be held in Las Palmas de Gran Canaria, SPAIN, from August 31 to September 4, 1992.

Keywords: Weightless Neurons, Boolean Networks, RAM-Based, GSN

Corresponding Author:

Dr. Edson Costa de Barros Carvalho Filho
Universidade Federal de Pernambuco
Departamento de Informática
Av. Prof. Luís Freire
Recife — PE — Brazil
CEP 50.739

E-mail: edson@di.ufpe.br

Weightless Neuron Models †

Edson Costa de Barros Carvalho Filho
Universidade Federal de Pernambuco
Departamento de Informática
Recife — PE — Brazil
CEP 50.739
e-mail:edson@di.ufpe.br

Abstract

The study of connectionist processing and artificial neural networks has generated a wide diversity of models, both in terms of the nature of the computational nodes employed and the structural architecture for implementation. While many models are based on nodes which compute the familiar sum-of-products function, and alternative class of neural networks use adaptive Boolean logic elements for the processing node. In principle, such networks offer a number of advantages relating particularly to their potential for high-speed processing, simplicity of implementation in VLSI, tractability of analytical studies and real world practical tasks. This class of nodes is known as weightless or RAM-Based neurons because they present a structure similar to a look-up table or a Random-Access-Memory. This paper review the Random-Access-Memory(RAM) Model[1], the Probabilistic Logic Neuron Model(PLN)[5], the Multi-Valued PLN[7] and the Goal-Seeking Model(GSN)[4].

Keywords: Weightless Neurons, Boolean Networks, RAM-Based, GSN

1 RAM Neurons

Motivated by difficulties in implementing McCulloch-Pitts neurons[6] and the interesting properties of Boolean nodes, Aleksander[1] designed the the special-purpose SLAM(Stored Logic Adaptive Microcircuit) device, later based on RAM(Random Access Memory) elements, as an electronic device capable of providing hardware implementation of universal sets of Boolean functions.

The RAM neuron accepts and produces binary values. A RAM neuron is a device, as illustrated in Figure 1, capable of computing any Boolean function with a given number of inputs. A RAM neuron can be described as consisting of a *set of input terminals* which repre-

† This work has been supported by the Brazilian Research Council(CNPq) and the Research Foundation of the State of Pernambuco(FACEPE).

sents the input of the neuron; a *set of weighted powers* which encode inputs in the form of addresses(the weighted powers never change and are used only to label input terminals), a *set of cells* which store the contents or learned information; an *output terminal* which returns an addressed content given by the input terminals; a *teach terminal* which provides the desired output; and a *operation mode terminal* which indicates if the neuron is in the learning or recalling phase.

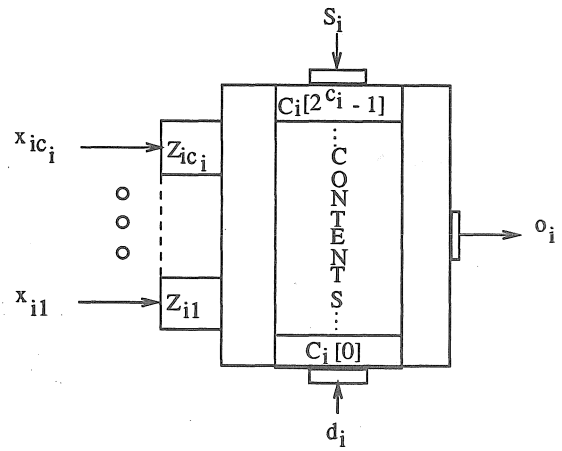


Figure 1: The RAM Model

A RAM neuron, n_i, with c_i input terminals has 2^{c_i} addressed contents and requires 2^{c_i} bits to be implemented. As all $2^{2^{c_i}}$ functions are computable, RAM neurons do not generalise. A RAM neuron has two states or operational modes: *the learning state* in which the neuron assigns the value on the teach terminal to the addressed content, and *the recall state* in which the neuron puts the addressed content on the output terminal. The output function is described by Equation 1.

$$o_i = \begin{cases} 0 & \text{iff } C_i[a_{i_m}] = 0 \\ 1 & \text{iff } C_i[a_{i_m}] = 1 \end{cases} \tag{1}$$

where $C_i[a_{i_m}]$ is the content of the address a_{i_m};

If the weighted powers are labelled as 2^k where $0 \leq k \leq c_i - 1$, then the address, a_{i_m}, accessed by an input, $x_i = x_{i1}, x_{i2}, ..., x_{ic_i}$, is defined by Equation 2

$$a_{i_m} = \sum_{j=1}^{j=c_i} x_{ij} z_{ij} \tag{2}$$

where x_{ij} is the input value of the j^{Th}-terminal and z_{ij} is the weight of the j^{Th}-terminal.

There are three major differences between the RAM and McCulloch-Pitts-like neurons:

1. McCulloch-Pitts like-neurons can have inputs, outputs and weights assuming any real number values, while the RAM neuron is essentially discrete;

2. The adaptive process in the McCulloch-Pitts neuron is related to the strength of the connections, while in the RAM model it is related to the addressable contents;

3. The McCulloch-Pitts model generalises in the sense that only linearly separable functions are computable, while the RAM neuron does not generalise, and all functions are computable.

2 Probabilistic Logic Neurons

After attempting a rapprochement between Boltzmann machines and logic node networks[2], in which the probabilistic nature of nodes is claimed to be important in solving pattern completion tasks, Kan and Aleksander proposed the Probabilistic Logic Neuron(PLN)[5].

The PLN model is a probabilistic version of the RAM neuron. The main difference between PLNs and RAMs is the inclusion of a third logic value such that when addressed produces a probabilistic binary output. This third logic value is equivalent to a "don't care", unknown or undefined value and is generally labelled 'u'.

The probabilistic output function that generates the neuron's output can be defined to produce zeros or ones with any probability, but it is normally considered to be the probabilistic function with equal likelihood of generating zeros and ones as defined by Equation 3.

$$o_i = \begin{cases} 0 & \text{iff } C_i[a_{i_m}] = 0 \\ 1 & \text{iff } C_i[a_{i_m}] = 1 \\ Ran(0,1) & \text{iff } C_i[a_{i_m}] = u \end{cases} \tag{3}$$

where $C_i[a_{i_m}]$ is the content of the address a_{i_m} and $Ran(0,1)$ is a random function that generates zeros and ones with the same probability.

In terms of information theory, the inclusion of the undefined value duplicates the memory requirements from 2^{c_i} bits for a neuron with c_i inputs, to 2×2^{c_i}.

The undefined value in a cell is used to identify untrained inputs, so if an untrained input appears and addresses an undefined cell the neuron simply produces a random output that is claimed as a form of internal generalisation. At the neuron level, the internal generalisation in the recall phase is totally independent of the learning experience since the random process does not consider past experiences(It is random). Even without training, the same untrained input can cause a PLN neuron to produce opposite output values, and this reflects the proba-

bilistic behaviour of PLN neurons. Some particular problems associated with the probabilistic behaviour of PLN neurons are discussed in [3].

3 Multiple-valued Probabilistic Logic Neurons

A cell in the PLN model is able to store zero, one or the undefined value. The probabilistic output function produces a zero or one with equal probability when the u value is addressed. In the Multiple-valued Probabilistic Logic Neuron(MPLM) proposed by Myers and Aleksander[7], a cell stores a range of values between 0 to 1, and the probabilistic output functions accepts the stored content as the probability of producing a 1 value. Stored content values vary from the complete randomness of the output at 0.5 to the absolute certainty of the output for the extreme values 0 and 1. There are two important parameters involved in the MPLM model, which are:

1. *The number of elements representing possible stored values* which is defined by the number of bits used in cells. After a punish/reward learning, the cells tend to acquire a zero or a one value. If a large number of elements is used to represent the probability of outputs, the learning process will lead showly to the internal saturation(evolution) towards one of the extremes. This is valuable when changes in the stored content are going in the right direction, however it can be undesirable when it is going in the wrong way. Also, there is a memory penalty to represent a cell as a word of B_l bits, that is $B_l 2^{c_i}$ as well as longer convergence times. If a smaller number of storage elements is used, as with the PLN neuron which only stores three values, the main problem is that a cell gets quickly saturated.

2. *The output function* which is applied on the addressed content. The output function may be a probabilistic, linear, step, or sigmoid function as described for the McCulloch-Pitts neuron. Some experimental results[7] has investigated the application of the sigmoid function, $\Phi(C_i[a_{i_m}])$ defined by Equation 4.

$$\Phi(C_i[a_{i_m}]) = \frac{1}{1 + \epsilon^{\alpha(-2C_i[a_{i_m}]+1)}} \tag{4}$$

where $C_i[a_{i_m}]$ is the content addressed by a_{i_m}, and α has been tested for values, 2.5, 4, 5, 10 and 25.

Clearly, the choice of the number of possible stored values and the output function depends on the problem involved in the task as well as other parameters such as: neuron connectivity,

training set size, number of neurons, desired output code, expected performance, order of presentation, etc. This problem is not simple and may involve a lot of simulation.

A MPLN net is said to converge when all locations in all nodes either have probabilities of outputting 1 equal to 0 or 1, or else are never addressed. After convergence, a MPLN behaves in exactly the same way as the PLN, and MPLNs could be replaced by PLNs preserving the behaviour and saving memory requirements[7]. This idea is not very attractive because a network should not be seen as static in the sense that after the learning phase nothing will change anymore.

4 Goal-Seeking Neurons

The GSN model accepts and generates values from the set $T = \{0, 1, u\}$, where the presence of undefined values on the input terminals identifies a set of possible addresses rather than a single address. The set of possible addresses is the *addressable set* and the set of contents thus addressed is designated the *addressable contents*. The area over which the goal is sought can therefore vary from a single address (in the case where all inputs are binary values) to the entire set of addresses (in the case where all inputs are undefined values). Computation is then accomplished by examining the addressable contents and generating the best value for the sought goal. The addressable set is determined by computing the fixed address given by the defined values on the input terminals and finding all combinations of addresses for the undefined values on the input terminals. A schematic representation of a GSN n_i is shown in Figure 2. In contrast to the RAM model, a GSN has a set of *state terminals* which indicate the state of the neuron and a set of *desired inputs* define the input signals which access the

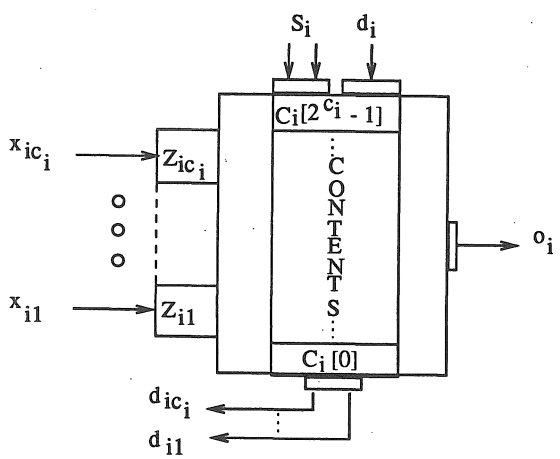


Figure 2: The Goal Seeking Neuron Model

The fixed address a_i^f of a neuron n_i is given by

$$a_i^f = \sum_{j=1}^{j=c_i} x_{ij} z_{ij} \mid x_{ij} \neq u; \tag{5}$$

where x_{ij} is the input value of the j^{Th}-terminal and z_{ij} is the weighted power value of the j^{Th}-terminal. Equation 5 computes the fixed address for inputs with defined values. Let $Z = \{z_{i1}^u, z_{i2}^u, ..., z_{iu_i}^u\}$ be the set of weighted powers with u_i-undefined inputs to n_i, thus the addressable set, A_i, is given by

$$A_i = \{a_{i_m} = a_i^f + \sum_{j=1}^{j=u_i} b_j^m z_{ij}^u, \forall m = 0, 1, \ldots, 2^{u_i} - 1\}; \tag{6}$$

where b_j^m is the j^{th} bit of the binary representation, $b^m = b_1^m b_2^m \ldots b_{u_i}^m$, of m. Equation 6 gives the 2^{u_i} addresses which make up the addressable set. Each address a_{i_m} represents a combination of binary values for the undefined input terminals, and an address a_{im} is computed by clamping the binary representation of m into the input terminals of undefined values, thereby defining all inputs and so calculating the address.

To complete the definition of the GSN, consider a specification of its goals at each state of operation as follows:

4.1 The Validating State

When a neuron is in the validating state the goal is to establish an undefined value, ensuring that the neuron can learn any desired output, otherwise the neuron can produce only a specific output during learning. The output in the validating state of a neuron n_i is given by

$$o_i = \begin{cases} 0 & \text{iff } \forall a_{i_m} \in A_i, C_i[a_{i_m}] = 0; \\ 1 & \text{iff } \forall a_{i_m} \in A_i, C_i[a_{i_m}] = 1; \\ u & \text{iff } \exists a_{i_m} \in A_i \mid C_i[a_{i_m}] = u \text{ or } \exists a_{i_m}, a_{i_l} \in A_i \mid C_i[a_{i_m}] \neq C_i[a_{i_l}]; \end{cases} \tag{7}$$

where $C_i[a_{i_m}]$ $C_i[a_{i_l}]$ are contents of n_i addressed by a_{i_m} and a_{i_l} respectively;

Equation 7 shows that when the addressable contents contain an undefined value, or when there is a mixture of 0 and 1 values then the output is undefined. The output is zero when all addressable contents are zero. The output is one when all addressable contents are one. This mode of operation helps to propagate undefined paths through the network, seeking out unused cells or conflicting values, indicating the possibility of learning a desired output. An unused cell can be used to store the desired output(0 or 1), and if there are conflicting values, then a cell that store the desired output bit is selected to be used in the mapping without disrupting previously stored information.

4.2 The Learning State

When the neuron is in the learning state the goal is to learn the desired output with an address which already stores an appropriate value, otherwise an undefined content is used to store the desired output. The sought address of a neuron n_i with desired output d_i is given by

$$a_{i_m} = \begin{cases} Ran(A_{i/_{d_i}}) & \text{iff } \|A_{i/_{d_i}}\| > 0; \\ Ran(A_{i/_u}) & \text{iff } \|A_{i/_{d_i}}\| = 0; \end{cases} \tag{8}$$

where $A_{i/_s}$ is the addressable set of n_i that stores the value s, and is given by $A_{i/_s} = \{a_{i_k} \in A_i \mid C_i[a_{i_k}] = s\}$, $Ran(A_{i/_s})$ is a random element of $A_{i/_s}$ and $\|A_{i/_s}\|$ is the number of elements in $A_{i/_s}$;

The neuron tries to associate the desired output with an existing cell in the addressable set. If the neuron fails, it chooses an undefined cell and sets its value to the desired output. If there are a number of possible choices a random decision is made. It is necessary to choose one particular cell to represent the output value because the address of this cell is passed back down the input connections as the desired output value for the previous layers. The desired inputs d_{ij}, which indicate the input required to recall the sought address is given by

$$d_{ij} = x_j \mid \sum_{j=1}^{j=c_i} x_j z_{ij} = a_{i_m};$$

Thus each neuron n_j receives its desired output, learn and provide the desired outputs to the previous layers.

4.3 The recall state

When the neuron lies in the recall state the goal is to produce the binary content value with the highest occurrence in the addressable set, and the output in the recall state is given by

$$o_i = \begin{cases} 0 & \text{iff } \|A_{i/_0}\| > \|A_{i/_1}\|; \\ 1 & \text{iff } \|A_{i/_1}\| > \|A_{i/_0}\|; \\ u & \text{iff } \|A_{i/_0}\| = \|A_{i/_1}\|; \end{cases} \tag{9}$$

If the number of ones in the addressable contents is greater than the number of zeros, then the neuron outputs a 1 value. If the number of zeros is greater then the neuron will output a 0 value. If the numbers of ones and zeros is equal then the neuron outputs an undefined value. Thus, even if the addressable contents contain only a single 1 value, and the remainder are undefined, the neuron will output a 1 value. The reason for adopting this scheme is to minimise the propagation of undefined values.

Naturally, the best goal for the GSN is problem-dependent, and the goals previously specified can be redefined appropriately, hereby creating a family of GSNs. Three different GSNs have been identified, as follows: the GSN for the feed-forward architecture previously defined, the GSN^S for self-organisation architectures, and the GSN^+ for feed-forward architectures. This is not to say that the GSN is only applied in a feed-forward architecture, but it is clear that by changing the goals it is possible to get more suitable behaviour for different architectures.

5 Conclusions

The basic processing mechanisms of weightless neurons has been investigated where each neuron model was completely defined by the activation function and operation modes. Clearly, there is an evolution in the complexity of the models which comes to tackle real world problems.

References

[1] I Aleksander. Adaptive systems of logic networks and binary memories. *Proc. Spring Joint Computer Conference*, 1967.

[2] I Aleksander. Adaptive pattern recognition systems and boltzmann machines: A rapprochement. *Pattern Recognition Letters*, 6:113–120, 1987.

[3] E C D B C Filho, D L Bisset, and M C Fairhurst. A goal seeking neuron for Boolean neural networks. In *Proc. International Neural Networks Conference*, volume 2, pages 894–897, Paris, France, July 1990. IEEE.

[4] E C D B C Filho, M C Fairhurst, and D L Bisset. Adaptive pattern recognition using goal-seeking neurons. *Pattern Recognition Letters*, 12:131–138, March 1991.

[5] W K Kan and I Aleksander. A probabilistic logic neuron network for associative learning. In *Proc. IEEE First International Conference on Neural Networks*, volume II, pages 541–548, San Diego, California, June 1987. IEEE.

[6] W S McCulloch and W H Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull Math Biophys*, 5:115–133, 1943. formal neuron.

[7] C E Myers. Output functions for probabilistic logic nodes. In *First IEE International Conference on Artificial Neural Networks*, pages 310–314, London, UK, October 1989. IEE.

Uma Linguagem Pseudo-Natural para Construção de Interfaces Multimídia

Mirela Gico Casado Ana Carolina Salgado

Departamento de Informática - UFPE
Av. Prof. Luiz Freire S/N
Cidade Universitária - Caixa Postal 7851 - Recife - PE
CEP 50739
e-mails: mgc@di.ufpe.br & acs@di.ufpe.br

Abstract

This work proposes an interface definition by users using for that a pseudo-natural language, PNAT. It includes the interpreter and its components : lexical analyzer, syntactical analyzer and executor. It is presented the interface system formalization to make uniform the interaction environment. The goal of the PNAT language is to facilitate the human-machine interaction since the interface manipulation becomes more pleasant and simple.

Key-words

linguagem natural, linguagem pseudo-natural, parsing, interação homem-computador

1 Introdução

Na maioria dos sistemas de banco de dados atuais, há uma crescente preocupação na melhoria da interface com os usuários. Dentre os vários tipos de interação homem-máquina existentes, menus, manipulação direta, linguagens de comando [Shné87], uma das formas mais humanas de manisfestação externa da atividade mental é o uso da Linguagem Natural (sendo um tipo de linguagem de comando). É a maneira que exprime mais facilmente idéias e desejos do ser humano.

Na comunicação homem-máquina o uso de linguagem natural requer a consideração de vários aspectos. Um dos grandes problemas é o da implementação de alguma forma de interpretação da linguagem, já que na nossa língua, uma mesma frase pode ter vários sentidos, ou mesmo várias frases podem querer expressar a mesma idéia. Não há portanto ainda um sistema que se possa definir como um sistema dito *completo*. Na realidade o que se tem são *sistemas simplificados* [Arab89], que utilizam subconjuntos de Linguagem Natural. Para tentar facilitar o acesso dos usuários ao sistema é definida uma *Linguagem Pseudo-natural*, a *PNAT*.

Em sistemas de Banco de Dados uma forma de fornecer ao usuário acesso às informações armazenadas é permitir a estes usuários uma maneira mais natural de expressar suas consultas. A linguagem definida, *PNAT*, é usada não como uma linguagem de consulta às informações do Banco de Dados, como na maioria das propostas de utilização de Linguagens Naturais ([Luk 86], por exemplo), mas sim para construir o ambiente de consulta. Para tal são utilizados conceitos do Sistema de Interfaces definido em [Salg90], que trata da manipulação de objetos complexos, usando uma estrutura denominada *cenário*.

Na próxima seção são apresentados os principais conceitos relacionados à interface. Na seção 3, é dada uma visão geral sobre Processamento em Linguagem Natural, sua importância, usos e problemas relacionados. A seção 4, apresenta a definição da linguagem PNAT. Finalmente, na última seção é apresentada uma breve conclusão e algumas observações sobre o trabalho.

2 Especificação da Interface

A interface é representada por uma estrutura complexa, que denominamos *cenário*[1] [Salg88], um novo conceito de interface de comunicação com o usuário, similar ao de "story" do sistema Storyboard da IBM [IBM 87] e ao de "card" da Hypercard da Aplle [Appl88]. Toda interação com o usuário é realizada através de cenários que podem ser pré-definidos pelo sistema ou definidos pelo próprio usuário. Estes cenários são apresentados através de objetos complexos (objetos multimídia [Fium87] e a maneira como o usuário define estes cenários é o ponto de interesse deste trabalho.

Antes de definirmos formalmente o ambiente de interface, vamos definir as diferentes classes de objetos a ele relacionadas : *cenários, cenas, mensagens, regras e restrições.* Um cenário é um conjunto de cenas às quais são associadas mensagens. A idéia de base é a ativação de cenas representadas no espaco ao longo do tempo. Um cenário, sendo uma forma de diálogo, é portanto dinâmico e necessita considerar informações tanto de tempo para a interpretação das necessidades do usuário, como de espaço para a apresentação dos diálogos.

Definição 1 : Restrições de tempo e de espaço (*RTE*)
Uma restrição de tempo e de espaco é um par : $RTE = <D, P>$

onde : D e P indicam respectivamente uma duração e uma posição no espaço de representação.

Definição 2 : Mensagem (MSG)
Uma mensagem é definida pela quádrupla : $MSG = <M, V, RTEm, T>$

onde M : método associado à mensagem,
V : valor resultante da execução do método,
$RTEm$: restrições de tempo e de espaço associadas a mensagem : D e P
Tp : temporização em relação ao começo da interpretação da cena a qual a

[1] O termo cenário é totalmente independente do conceito de cenário usado em aspectos específicos de Inteligência Artificial.

mensagem está associada. Esta noção permite definir a ordem de ativação das mensagens em uma cena.

Definição 3 : CENA
Uma cena é uma tripla : $CENA = < MSG, RTEc, R>$

onde $RTEc$: são as restrições de tempo e espaco associadas à cena: D e P
R : é a representação da cena para o usuário.

Definição 4 : REGRA
Uma *REGRA* pode ser definida pela regra de produção:
$REGRA = <$ Se $COND$ Então $MSG >$

onde $COND$: Corresponde a uma expressão lógica que associa uma ou mais cenas e que, se for verdadeira, ativará a MSG relacionada.

Podemos agora definir formalmente o conceito de cenário.
Definição 5 : CENÁRIO
Um cenário é o par : $CENÁRIO = <$ *cenas, regras* $>$

onde $\{cenas\}$: é o conjunto de cenas que compõem o cenário;
$\{regras\}$: define o agrupamento das cenas a ser considerado no momento da interpretação do cenário.

A interpretação de uma cena pode gerar, por exemplo, a visualização de uma imagem ou seqüência de imagens em um monitor, a emissão de uma informação de audio, a execução de um programa, a edição de um texto, ..., caracterizando assim um ambiente multimídia.

O nosso modelo de dados, pode ser representado como um objeto complexo com um conjunto de atributos [Bane88]. Uma classe pode ser definida como um domínio de um atributo. Este domínio, a menos que seja um domínio simples (D, P, M, V, ...), consiste de um conjunto de atributos, onde cada domínio de cada atributo corresponde a uma nova classe e assim por diante. Se o domínio de um atributo é um conjunto de uma classe, o valor deste atributo para qualquer instância será um conjunto de objetos (Fig. 1).

3 Processamento em Linguagem Natural

Processamento em Linguagem Natural (PLN) é a área da Inteligência Artificial voltada à geração automática e compreensão da linguagem humana.

Há boas razões para considerar o uso destas linguagens em computadores. A primeira razão é que as informações a serem manipuladas normalmente estão em linguagem natural. A segunda razão é que as pessoas necessitam para se comunicar com as máquinas de linguagens mais "naturais", já que estas facilitam sua comunicação.

Infelizmente, embora o mapeamento entre a linguagem como um todo e o mundo considerado seja natural, o mapeamento entre um modelo e a parte correspondente da linguagem

não é tão natural. Sem dúvida, um dos maiores problemas no Processamento de Linguagem Natural é o da ambigüidade, isto é, a existência de várias formas de entendimento de uma mesma frase. Nos Sistemas de PLN o maior problema é a transformação de uma frase potencialmente ambígua em uma não ambígua que será utilizada pelo sistema. Esta transformação é conhecida como "Parsing".

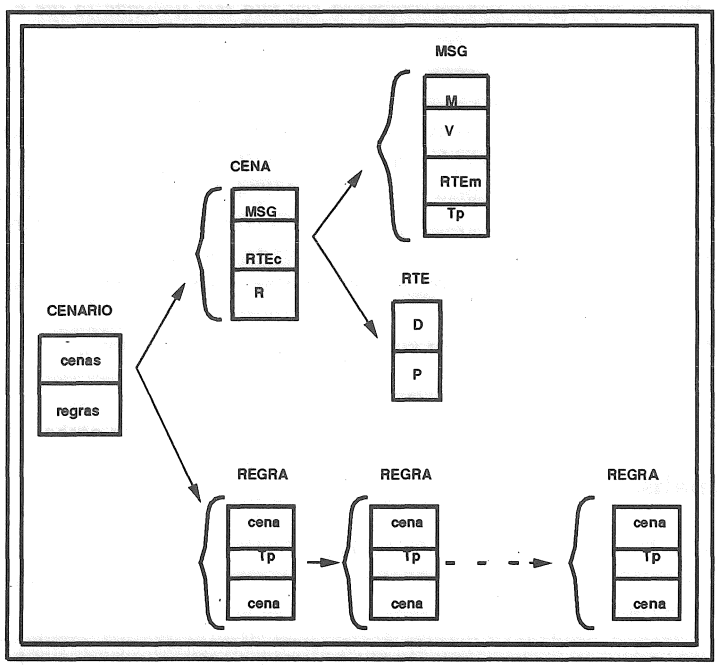


Fig 1 Classes de objetos da Interface

Existem cinco maneiras de se realizar o "parsing" : Pattern-Matching [Weiz66], Baseada em Gramáticas [Hopc79], Semântica, Baseada em Conhecimento e Redes Neurais.

4 PNAT : uma Linguagem Pseudo-Natural

A PNAT é uma linguagem pseudo-natural que objetiva facilitar a interação do usuário com a máquina, permitindo que estes construam seus próprios ambientes de trabalho. Normalmente o que se tem é a possibilidade de acesso às consultas ao Banco de Dados utilizando LN.

Como qualquer linguagem, a PNAT, necessita ser interpretada para que possa ser executada. É necessária então a definição de um *interpretador*. São utilizadas para a sua

interpretação duas técnicas : "Parsing" baseado em gramática e Pattern-Matching. Este interpretador é composto de dois analisadores e um módulo executar.

Os analisadores são: analisador léxico e analisador sintático. Para a análise sintática, foi escolhida uma gramática livre de contexto ao invés de uma sensível ao contexto porque não foram consideradas restrições rígidas para concordâncias nominal e verbal. A análise não é feita rigidamente, isto é, tanto faz o usuário digitar *a interface* como *as interface*. Não é considerado este tipo de erro gramatical. Assume-se que existem palavras-chave que são os objetos e os comandos de manipulação da interface. Na gramática são consideradas algumas palavras sinônimas, para permitir ao usuário maior flexibilidade.

O módulo executor utiliza a técnica de Pattern-matching. Ele associa a cada palavra-chave encontrada, no conjunto gerado pelos analisadores na etapa anterior, ações que ele deverá realizar. Estas ações são as necessárias à manipulação da interface.

A interpretação é realizada sobre cada unidade elementar de comunicação que são as *frases* [Luft74]. A interpretação gera a transformação da frase do usuário em uma frase a ser entendida internamente pela máquina e que ao mesmo tempo traduza a mesma intenção da frase do usuário. Nota-se que como a preocupação neste trabalho se concentra na construção de interfaces, a gramática da linguagem se restringe apenas aos comandos necessários para construção, manipulação e remoção destas, desprezando as frases que se destinem a consultas.

São descritos a seguir os módulos componentes do Interpretador.

4.1 Analisador Léxico

O Analisador Léxico tem a função de receber a frase do usuário e transformá-la em uma lista de palavras, para que possa ser processada pelo Analisador Sintático. Uma frase acaba sempre com um ponto.

Em uma frase a unidade elementar é a *palavra*. Uma palavra é considerada como um conjunto de caracteres, que incluem letras maúsculas e minúsculas, os números arábicos e todos os caracteres de pontuação (, . ? ! " ...). Sabe-se que chegou ao fim de uma palavra quando encontra-se o caracter " " (branco). Existem na PNAT *palavras-chave* da linguagem e quando é realizada a Análise Léxica, é feita a uniformização e retirada de porções supérfluas das palavras-chave. O critério utilizado para a escolha do que seriam estas partes supérfluas das palavras adveio da percepção da menor porção da palavra, isto é, dos caracteres mínimos que identifiquem o objeto ou o comando mencionado.

Por exemplo, na frase : (*Construir Interface :ajuda*) o Analisador Léxico devolve então a seguinte lista de palavras : *CONST, INTER, :AJUDA*. Houve, por exemplo, a supressão na palavra *Construir* da porção *ruir,* e todas as palavras foram colocadas em maiúsculo integralmente. Tanto faz o usuário escrever : *Construir Interface :ajuda* ou *construiremos Interface ajuda* ou qualquer tempo verbal ou inflexão pois as porções "a mais" são desprezadas. Na verdade, só são analisadas as frases que visem a manipulação de interfaces. As demais não são tratadas.

4.2 Analisador Sintático

Este módulo é o responsável pela verificação se o formato (sintaxe) dos comandos é válido. A *sintaxe de uma linguagem* é o estudo das regras que determinam quais cadeias de palavras podem formar frases [Sava88].

Foi escolhida uma gramática Livre de Contexto para geração da PNAT. Este tipo de gramática é muito utilizado na análise de Linguagens de Programação pela sua simplicidade e eficiência dos analisadores obtidos. Por conseguinte também é muito usada na análise de Linguagens Pseudo-Naturais.

As palavras apresentadas em negrito na gramática (Fig 2), representam os símbolos terminais, isto é, as palavras-chave. As porções grifadas são suficientes para que as palavras sejam reconhecidas pelo Analisador Léxico. Todos os objetos que aparecem entre *[]* (colchetes), são opcionais na emissão do comando. Se forem omitidos, o sistema se encarregará de questioná-los ao usuário. Nota-se que há também a presença de alguns sinônimos (ALTERAR E MODIFICAR, CONSTRUIR E FAZER, MOSTRAR E APRESENTAR, ...) com o intuito de permitir ao usuário mais opções no momento da interação.

Como visto na seção 2, a interface é composta de cenários e estes por sua vez são compostos de cenas. Quando um usuário requisita a construção de uma cenário, o sistema deve ir descendo até o nível mais baixo da *árvore de construção de interface.* A árvore de construção de interface *(ACI)* corresponde à árvore de hierarquia entre os objetos componentes de uma interface (Fig. 3). Descer até o nível mais baixo da árvore de hierarquia significa que o sistema interpela o usuário sobre os objetos do nível imediatamente inferior e continua descendo até que se chegue às folhas da árvore.

Nota-se que há sempre a ligação entre os objetos manipulados via identificadores que são colocados nos níveis inferiores da ACI, isto é, cada cena, por exemplo, possui na sua construção a identificação do cenário do qual faz parte. Deve-se notar também que alguns componentes dos objetos manipulados pela PNAT foram omitidos das produções da gramática. Um exemplo é a quantidade de cenários componentes de uma interface, as restrições associadas às cenas, as posições de representação das restrições de tempo e espaço. Todos estes elementos são requisitados ao usuário pelo sistema no momento da construção do objeto que o contém.

4.3 Executor

Este módulo é o encarregado de executar as frases que já foram verificadas pelos módulos de análise. Este módulo só é ativado se a frase conseguiu passar pela análise sintática e léxica

Neste ponto as porções supérfluas das palavras já foram retiradas e o que se tem é um conjunto de palavras-chave e de identificadores. Cada identificador vem precedido de dois pontos (:).

A cada palavra-chave está associado um conjunto de ações (rotinas) a serem realizadas. O método utilizado é o de Pattern-Matching.

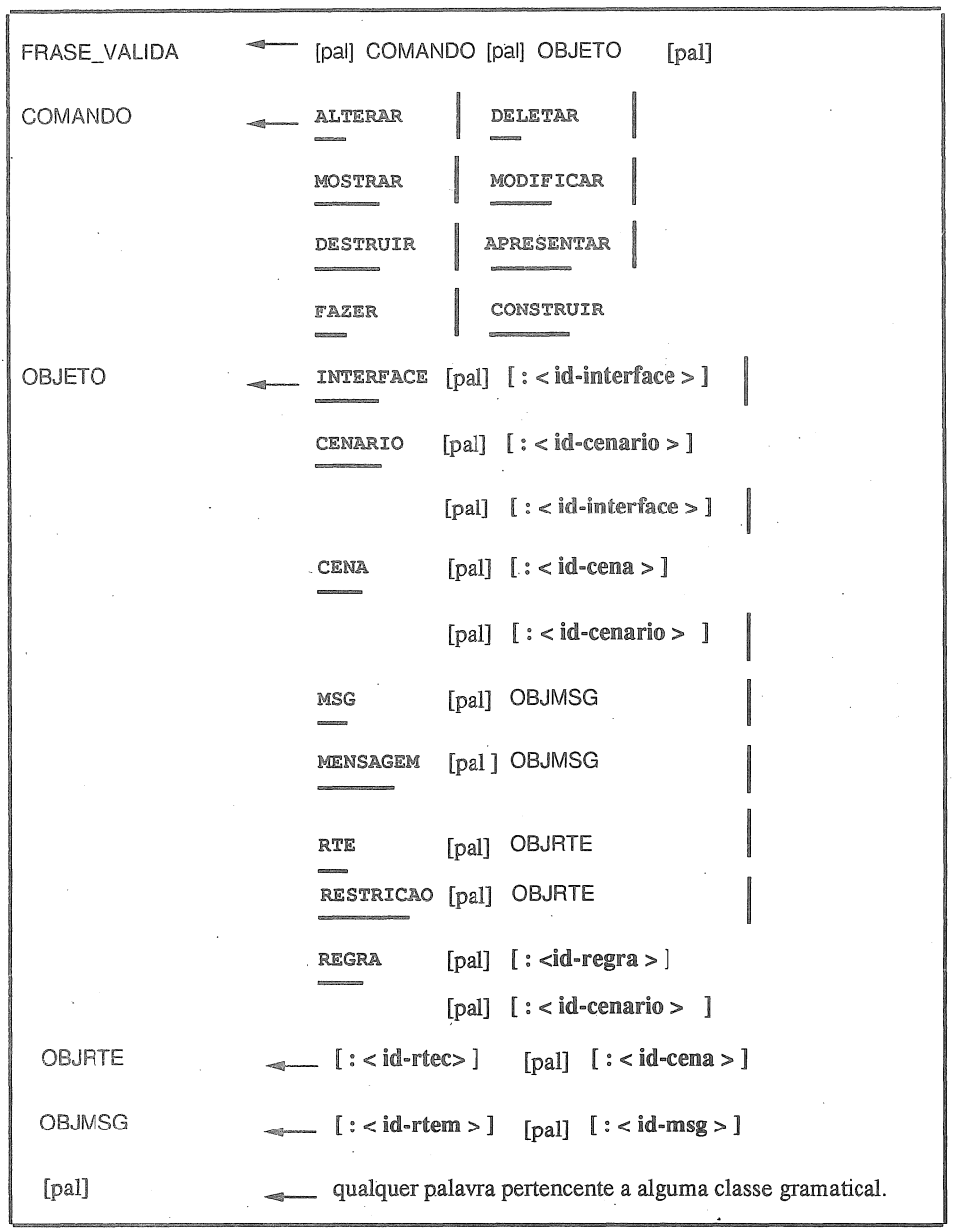


Fig 2 Gramática da PNAT

Para que os comandos sejam entendidos corretamente, é necessário apenas que alguma

parte ou partes dos comandos sejam escritos, porém a inclusão de palavras que não fazem parte da sintaxe da linguagem tais como artigos, adjetivos, conjunções preposições etc., ou mesmo a mudança do tempo verbal dos comandos, não influi no "entendimento" por parte do Executor, já que estas foram retiradas pelo Analisador Léxico.

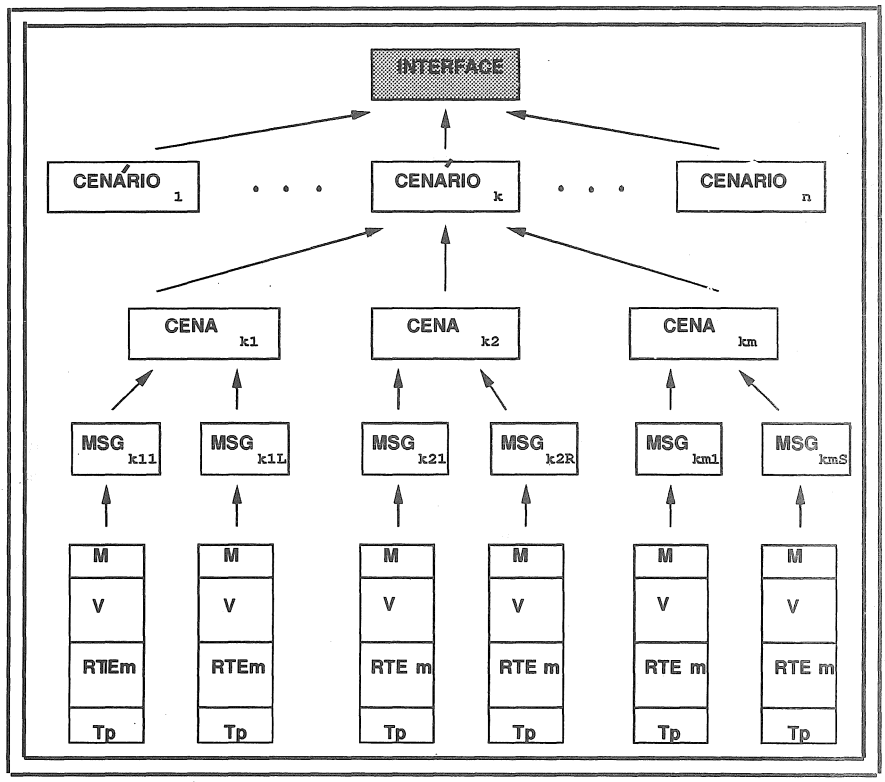


Fig 3 Árvore de Construção de Interface

As seguintes frases :

- *CONSTRUIR INTERFACE* :I1
- *CONSTRUA INTER* :I1
- *CONSTRUA* UMA *INTERFACE* :I1

teriam o mesmo efeito de execução isto é, seria contruída a interface I1. O sistema se encarrega de questionar ao usuário sobre a quantidade de cenários associados a esta.

Para os comandos de alteração e remoção é suficiente que o usuário especifique qual objeto (Cenário, Cena, ...) que deseja alterar ou remover indicando ou não o seu identificador. Se o identificador não é indicado, este é questionado ao usuário. O usuário deve ao menos conhecer em qual interface está o objeto a ser alterado ou removido. Ao ser identificado, a estrutura do objeto é apresentada ao usuário que pode então ter acesso às informações que deseja alterar. Se o objeto a ser deletado não for uma folha na ACI, será solicitado ao usuário uma confirmação da remoção das estruturas abaixo deste objeto.

O comando MOSTRE proporciona ao usuário a visualização da ACI de uma dada interface a partir do objeto desejado, caso seja especificado o id-obj, ou de um determinado nível da ACI, caso o id-obj não seja especificado. No segundo caso, todos os objetos são apresentados um de cada vez (as sub-árvores). Vale frisar que o usuário deve especificar o identificador da interface que ele deseja visualizar.

5 Conclusão

Atualmente não existe ainda uma linguagem dita totalmente "natural" que atenda a todas as necessidades do usuário. O escopo de aplicação tem que ser restringido pela dificuldade que existe em expressar fielmente os objetivos dos usuários. Isto é conseqüência da inerente ambigüidade da linguagem natural. Na realidade o que existe atualmente são linguagens Pseudo-Naturais com objetivos próprios como em : [Jans90], [Oliv89], [Arab89], [Bier85], [Luk 86], [White90], entre outros trabalhos.

A linguagem Pseudo-Natural descrita facilita a interação com o usuário, permitindo-lhe construir as interfaces para as suas aplicações. O que a diferencia das demais linguagens Pseudo-Naturais, é que esta não é uma linguagem de consulta à base de dados, mas sim uma linguagem dedicada à contrução de interfaces. Utiliza para tal os conceitos de cenários, cenas, regras, etc., que permitem a uniformização das interfaces a serem construídas.

Referências

[Appl88] Hypercard User's Guide, Apple Computer, Inc., 1988.

[Arab89] ARABE A. M. F. : "Comunicando-se com Bases de Dados Estáticas em Linguagem Natural : O protótipo de uma interface", 6 Simpósio Brasileiro em Inteligência Artificial, 1989.

[Bane88] BANERJEE J., KIM W., KIM K.: "Queries in Object-Oriented Databases", IEEE, 1988.

[Barn90] BARNETT J., KNIGHT K., MANI I., RICH E., "Knowledge and Natural Language Processing", Communications of the ACM, Vol. 33, No. 8, 1990.

[Bier85] BIERMANN A.W., RODMAN R.D., RUBIN D.C., HEIDLAGE J.F., "Natural Language with Discrete Speech as a Mode for Human to Machine Communications", Communications of the ACM, Vol 28, No. 6, 1985.

[Fium87] FIUME E., TSICHRITZIS D. : "Multimedia Objects", IEEE, 1987.

[Harri78] HARRISON, M.A. :"Introduction to Formal Language Theory.", Addison-Wesley, 1978.

[Hirs88] HIRSCHBERG J., BALLARD W.B., HINDLE D., "Natural Language Processing", AT & T Technical Journal, Vol 67, 1988.

[Hopc79] HOPCROFT J. E., ULLMAN J. D., "Introduction to Automata Theory, Languages, and Computation", Addison-Wesley Publishing Company, 1979.

[IBM 87] IBM Storyboard Plus, Manual de utilização do sistema, IBM Corporation, Junho 1987.

[Jans90] JANSEN B., PROIX C., GROSZ G., ROLLAND C., "Using Object Concepts to Show the mapping between Knowledge in Quasi-Natural Language and Prolog", Institut Blaise Pascal-4, Place Jussieu, 1990.

[Luft74] C.P. LUFT. :"Moderna Gramática Brasileira", Editora Globo, 1974.

[Luk 86] LUK W. S., KLOSTER S., " ELFS : English Language From SQL", ACM TODS, Vol 11, No. 4, 1986.

[Ober87] OBERMEIER K. K., "Natural-Language Processing", Byte, 1987.

[O iv89] OLIVEIRA C.A. : "A Sintaxe, a semântica e a pragmática : um enfoque integrado baseado no conhecimento lingüístico textual ", 6 Simpósio Brasileiro em Inteligência Artificial, 1989.

[Salg88] SALGADO DE AGUIAR A.C. : "Contribution a un SGBD orienté objet (NICEBD) : traitement des données et des interfaces multimedia", Tese de doutorado, Universidade de Nice, Outubro 1988.

[Salg90] SALGADO DE AGUIAR A.C., FONSECA D., CASADO M. G. : "Uniformização de Interfaces de comunicação em um Ambiente Multimídia ", 5 SBBD, 1990.

[Sava88] SAVADOVSKY P. : "Introdução ao projeto de Interfaces em Linguagem Natural ", SID Informática, 1988.

[Shne87] SHNEIDERMAN B., " Designing the User Interface : Strategies for Effective Human-Computer Interaction", Addison-Wesley Publishing Company, 1987.

[Weiz66] Weizenbaum J., "ELIZA - A Computer Program For the Study of Natural Language Communication Between Man And Machine", Communications of the ACM, Vol 9, Num 1, 1966.

[Whit90] WHITE G.M., "Natural Language Understanding and Speech Recognition", Communications of the ACM, Vol. 33, No. 8, 1990.

Design And Cooperation Management Through Design Steps and Object Characteristics

Maria Aparecida Castro Livi & Cirano Iochpe

CPGCC-UFRGS
Caixa Postal 15064
91501 Porto Alegre RS
BRAZIL

e-mail: iochpe@inf.ufrgs.br

ABSTRACT: Conventional database management systems (DBMS) do not cope well with the requirement of cooperative work in the design environment (DE). They implement conventional, short transactions which run in isolation preventing designers from exchanging non-committed object updates. Cooperation models are being proposed which aim at supporting DE's dynamic properties (i.e. its processing model) especially with respect to both object-transfer operations and the definition of application-oriented correctness criteria for the database (DB).

This work gives an overview of the design management and cooperation model for CAD database systems being developed at the CPGCC-UFRGS and presents a mechanism which realizes it. The model supports object definition by means of application-oriented object characteristics (properties). Characteristics are integrated into objects through the execution of design steps. Design steps are related to one another forming partial orders which represent the design process. Relying on sets of object characteristics and partial orders of design steps, the DBMS's Design and Cooperation Manager (DCM) guarantees database correctness by controling the design process as well as object-transfer operations.

1. Introduction

Conventional DBMS implement the transaction paradigm [2] to support the application's dynamic properties. Application work must be modeled as (short) transactions which follow the ACID principles of atomicity, consistency, isolation, and durability [3]. In this context, database correctness can be guaranteed on the basis of serializable schedules of concurrent transactions [1]. Concurrency control techniques that rely on serializability need not take application semantics into consideration.

Processing models of design applications (e.g. VLSI design, software engineering) pose new requirements on the DBMS's transaction manager. Design work can last very long (e.g. days, months). From the application point of view, it is impossible to model it as an atomic transaction that must be completely undone in case of error or failure. Furthermore, the design work is a cooperative activity ([5], [6], [8]). Designers exchange information about their work in order to achieve common goals. Consequently, the requirement of isolation does not apply to design transactions.

The STAR environment for VLSI design [10] being developed at the CPGCC-UFRGS integrates a (long) design transaction manager [9] and a design and cooperation manager ([4], [6]). The latter implements an extension of the cooperation model proposed in [5]. It relies on sets of application-oriented object characteristics as well as partial orders of design steps to control the correctness of the design process.

After giving an overview of the design management and cooperation model (DMCM) proposed in [6], this paper presents the design and cooperation manager being implemented as part of the STAR project. The rest of the text is organized as follows. Section 2 discusses DMCM's main features. Section 3 presents the design and cooperation manager including its architecture and the information it works upon. Section 4 concludes the paper and points out to future work.

2. A Design Management and Cooperation Model

The model being developed at the CPGCC-UFRGS supports two views of the evolutionary character of design applications: object construction and design process development. Design objects are specified to the system by means of their (future) features which should be realized during the design process. The latter is modeled as a partial order of design steps. Each design step manipulates a set of objects and implements some of their (expected) features. Design steps incrementally change object states until objects reach their final states, that is, states in which objects incorporate all features declared in their specifications.

2.1 The Underlying Computer System Configuration

The model supposes a distributed computer system with a server-workstation architecture. Server nodes store application objects (e.g. designs, libraries, processing tools, and administration data). Designers work on private workstations connected to the server nodes through a communication subsystem (e.g. a LAN). Therefore, design work actually takes place at the workstation sites.

2.2 Transaction Management and Database Architecture

According to the hierarchical structure of design projects, the transaction model supports the specification of transaction hierarchies. A project transaction (TApj) represents the whole effort of a group of designers working on a common design. A project database (DBpj) is associated to every TApj at its start. When a project needs to access an application object that is not locally stored in its DBpj, the TApj issues a checkout operation in the public database (DBpu). Concurrent project transactions are isolated from one another through a, possibly non-strict, two-phase locking mechanism.

Each designer develops her/his work from inside a design transaction (TAd). Every TAd is related to a private database (DBpr). Since both design and project transactions are of long duration, they can be subdivided into sets of (possibly concurrent) ACID transactions.

Design transactions of the same project are considered to be nested transactions of the respective TApj. They cooperate with one another to carry out the design project. DMCM controls the information flow among design transactions.

The transaction and database hierarchies described above are mapped onto the computer network as follows. Project transactions execute at server nodes. Similarly, project databses and the DBpu are located at those processing nodes. Design transactions execute at workstation sites. Private databases are located there, too.

2.3 Object Specification and Object Design Evolution

The specification of a design object includes a set of features that represents the set of characteristics which must be present in the object at the end of the design process. Object features express design decisions about form, performance, functions, and correctness criteria. They can also represent the ocurrency of either a test or a simulation of an object. Moreover, the object specification can either establish or estimate values for the features (e.g. the area of a chip must end up between 2 and 3 cm^2). An Object specification can be updated at any time.

New features can be added to as well as existing features can be deleted from it. Designers can also change expected feature values at any time.

Object design evolution is captured by a partial order of object versions. Each version expresses one of the states which were already reached by the object due to the design process. Each state is represented by a specific set of (realized) object characteristics. Therefore, versions of a same object differ from one another either in the set of features they incorporate or in the values they set to them.

Object versions can assume three different development states:

- **in-work**: The version is kept in the private database of any of its creators and is not eligible for transfer operations. Therefore it can be deleted at any time without affecting other designers.
- **stable**: The version already incorporates a significant subset of its expected characteristics. Therefore, it can be lent other users to help them by their own designs. Stabel versions are kept in the project database.
- **consolidated**: The version presents all of its expected characteristics and its creators release it to public use (i.e. use in any project). It is transferred to the DBpu and cannot be deleted anymore.

Object specifications as well as object versions are given unique identifiers (surrogates). Therefore, there are three distinct ways to identify either a specific object version or a set of versions: through the unique identifier of the specific version; giving the id of an object specification together with a set of object characteristics (either related to specific values or not); giving only the id of an object specification to automatically retrieve its most actual version.

The use of object characteristics in the identification of versions enables designers to identify sets of versions that have the same subset of features or even object versions which have not yet been created. Both advantages can be explored by the cooperation manager to control the correctness of object-transfer operations (see below). Though, there are some disavantages in the use of object characteristics to identify versions. System implementation can be more complex and query processing can last longer.

2.4 Project Specification and Design Process Evolution

Usually, projects in areas such as VLSI design, software engineering, and mechanical parts design are very complex and tend to be recursively subdivided into subprojects until each design (sub)task achieves a complexity level that can be handled by a small group of specialists. DMCM views a design project as a hierarchy of design (sub)tasks (i.e. subprojects), each one of which develops one or more (sub)objects. The latter are integrated to form the final product of the design project (e.g. the design of a cpu could include the design of a main memory component, the one of a bank of registers, that of an ALU, and the bus design).

Each (sub)project is modeled by means of a set of object specifications and a project specification. The object specifications describe the tasks of the project. The project specification defines the way it should be carried out. A project specification consists of a partial order of design steps (DS). Each DS expresses a set of design activities which have a common meaning (e.g. form a specific design phase) for the application. During its execution, a design step can implement new object characteristics, test already existing ones, and inform designers about object design evolution.

An DS is responsible for the execution of a set of application-oriented design tools (e.g. editors, compilers, and simulators), each one of which accesses existing object versions and can generate new ones. The model supports human intervention to be represented as an application tool inside the step.

DMCM allows designers to define any execution order for the tools inside an DS. In addition, designers must point out the subset of object versions to be created inside a design step that

must be kept in the database at the end of its execution. The design and cooperation manager must pass this information on to the DBMS's version manager.

At DS definition time, designers must declare specific object versions as either input to or output of design tools. Relying on this information, DCM can infer the exact point in time when a specific version will be needed for cooperation. It can also infer when it will be produced as well as in which DS it will occur.

Not only its relative position in the partial order of design steps determines when an DS should be started. Designers can associate a start predicate to every design step. Such a predicate can relate the execution of its associated step to the realization of any object characteristic as well as to the evaluation of any query against the database. DCM can control design process evolution by keeping information about the directed graph of design steps as well as evaluating the start predicates of steps which are ready to execute.

DMCM allows step specifications to be updated at any time. Moreover, the model enables designers to suspend, resume, and abort step executions. Any of these operations on one DS can force DCM to abort or compensate for those design steps which depend upon that one and either executed already or are still executing.

2.5 Object-Transfer Operations

Cooperation among designers of a same project is supported by the design management and cooperation model on the basis of both object characteristics and object-transfer operations.

As mentioned above (see section 2.3), any designer can access object versions stored in the public database. Object versions in a DBpj can be accessed only by designers working on the respective project. Finally, versions stored in a private DB can be accessed by its owner only.

Since DMCM allows the design of an object to be carried out by more than one designer, the transfer of an object version between the private databases of two of its creators has not the same semantics as a transfer operation which lends a copy of an object version to a non-creator designer. While the prior operation prevents the first creator to contiue working with the version, the latter prevents the user of the version to update it, while the creator can continue designing the object (i.e. producing new versions of it). Therefore, transfer operations between creators transfer object versions from one DBpr to another. On the other hand, object-transfer operations between creator and user send copies of object versions from project databases to private ones.

Actually, the model supports two kinds of cooperation: between designers (i.e. creators of a same object) and between projects (creator-user relationship). The prior takes place inside the same project specification (i.e. partial order of design steps), every time a following design step is to be executed under the responsibility of another designer (possibly at another workstation). In this case, there is only one copy of an object version that is transferred from one database to the other.

Cooperation between projects implies the existence of, at least, two copies of a same version: one stored in a DBpj related to its creators and the others being used by non-creator designers in their own DBpr. DCM can detect when this kind of cooperation must occur by controlling the execution state of existing partial orders of design steps. Moreover, it can also notify creators of pending object-transfer operations so that they can eventually change the implementation order of some object characteristics.

Since new object versions can be created in parallel to the use of some older version of the same object, DCM must keep track of project cooperation. New versions should eventually substitute older ones at user sites. Object versions either composed by or whose design relies upon borrowed versions cannot be consolidated in the DBpu before those borrowed versions are. In addition, DCM must identify those design steps (of any existing project specification) which relied on any object version created inside a DS that has been aborted or compensated for. Responsible designers must decide whether affected design steps should be completely aborted or not.

The end of a cooperation at the project level is usually decided by designers responsible for the project specification that borrowed the object version(s). On the other hand, cooperation can either remain until the end of the whole project or be cancelled due to either DS abortion or version deletion.

3 A Mechanism for Design and Cooperation Management

This section discusses a mechanism which implements the design management and cooperation model described above. At present, a prototype of it is being developed as part of the STAR environment. The design and cooperation manager is being modeled on top of the KRISYS (Knowledge Representation System) [7]. Its routines are being written in LISP and C. We expect to have a running version of the DCM by the end of 1992.

To correctly support design process evolution, DCM must maintain information about object as well as project specifications. Since this information must be stored in the database but, at least in the present version of the mechanism, is not available to the application tools through the usual database interface, it is kept in a special repository called design and cooperation database (DBdc).

DBdc's conceptual schema is shown in Figure 1 as an E-R diagram where entities are drawn as rectangles and relationships as circles. The schema is known to the designers who create and update it during the whole design process. For this purpose, DCM supports operations to insert, update, delete, and query all information kept in the DBdc. DCM, too, updates the design and cooperation DB. It must keep track of state transitions on objects and steps as well as control both cooperation between designers and projects.

DCM works close together with two other STAR components: the transaction manager (which actually is being implemented as part of KRISYS) and the version manager. Partial orders of design steps are mapped onto (long) design transactions and the execution of application tools inside steps occur as short, ACID transactions. Only down to DCM's level, design steps are seen as application-oriented units of work. Object versions generated inside design steps are passed on to the version manager that builds (version) derivation graphs for each object specification at all levels of the database hierarchy.

Designers use DCM's services for two purposes: create as well as update DBdc and start as well as control the design process. In the following, on the basis of its data structures (see Figure 1) we discuss the main services to be implemented by the mechanism.

3.1 Defining Projects and Objects

DCM supports the partioning of projects into subprojects. This information is kept by the **projects** entity and the **subproject** relationship. Every project is related to a **design-specification** as well as a set of **object-specifications**.

Objects can be related to sub-objects through the **sub-object** relationship. Moreover, each object specification **incorporates** a set of **object-characteristics** and can be **represented-by** many different **versions**. DCM keeps information about version derivation by means of two relationships (to speed up retrieval): **derived-from** and **derives**.

Each design specification is related to a set of **design-steps**. The relationships **predecessor** and **successor** express the partial order of design steps. Each DS can be related to versions in three ways. It can **implement**, or be **waiting** for, or **borrow** object versions. While the first relationship is established at DS definition time, the other two ones are inserted into DBdc during the design process. Steps are related also to application tools (by the **activate** relationship).

Tools either **realize** or **test** (i.e. verify/simulate) object characteristics while creating new object versions. Versions can both **present** characteristics and set up values to them.

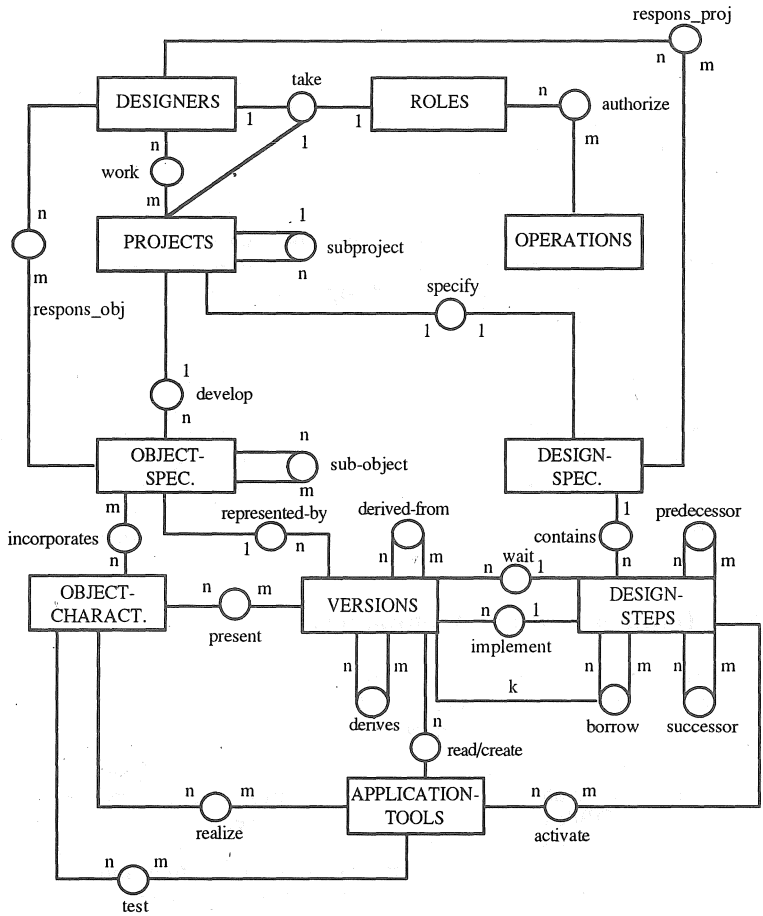


Figure 1: DBdc's conceptual schema

3.2 Giving the Designer a Role

A designer can **work** in one or more projects but in each one of them she/he can **take** only one **role**. The latter **authorizes** the subset of DCM **operations** which can be executed by designers related to it. Examples of operations are the creation of new projects, the insertion/deletion of an object characteristic, and the execution of a design step.

Depending on her/his role in the project, a designer can be responsible for the specification of one or more objects (**respons-obj**) as well as the design specification (i.e. definition of design steps and the relationships where they play a role). In addition, designers responsible for a design specification are the only ones who can start as well as control execution of its design steps.

3.3 Executing Partial Orders of Design Steps

By starting the execution of a design specification, the responsible designer may choose from one of two execution modes: automatic or monitored. The prior implies that DCM does not have to ask for permission as to start and commit the execution of design steps. It keeps

information about step dependencies and starts predicate evaluation to decide which DS (or set of design steps) can be started at each point in time. Monitored execution forces DCM to ask the responsible designer for permission every time a DS can be started. The latter form of execution enables designers to share the responsibility over the design process. Different steps can be executed at different workstation sites. Automatic execution, on the other hand, keeps on starting design steps at the site where the execution of the partial order began. Only in case of object fault in the actual DBpr, DCM halts step execution and asks the responsible designer for a decision.

During the execution of a partial order, DCM updates DBdc when necessary. Thus, if a design step inside which new object versions were generated commits, the design and cooperation manager updates the **version** as well as the **design-steps** entities and starts looking for pendent object-transfer operations which can use the just created versions.

If either an object or a design specification is updated during the execution of related partial orders of design steps, DCM investigates the consequences of such an act. For instance, in case a relationship between an object characteristic (Ci) and an object specification (Sj) is removed from **incorp**orates, DCM searches the tool which should realize Ci in Sj, identifies the versions of Sj which should have that characteristic and the design steps that either created or must create such versions. From this point on, DCM can identify what must be modified in the partial order (i.e. steps which must be aborted or compensated for) and which other partial orders will be affected by such modifications (i.e. the ones that contain steps which borrowed object versions from steps being aborted or compensated for).

3.4 DCM's Implementation Architecture

Two software modules implement STAR's design and cooperation manager: the project manager (PM) and the cooperation manager (CM). PM supports the managerial aspects of a design project while CM supports the cooperation among designers.

The project manager inserts and maintains all information kept in the design and cooperation database but those data related to cooperation activities. It guarantees design process correctness by enforcing DBdc consistency, controlling the execution of partial order of design steps, evaluating start predicates, identifying possible commit points for design specifications, mapping partial orders onto (long) design transactions, submitting application tools for execution as short, ACID transactions, and interacting with STAR's version manager to declare as well as access and delete object versions in the design database hierarchy.

The cooperation manager controls cooperation activities among designers by keeping track of object-transfer operations. It searches for versions being requested by design steps and verifies the possibility of lending them to the steps. CM maintains all information concerning borrowed object versions (i.e. creators, users, involved databases, predecessors, successors, creator step, user steps, etc.). Relying on these data, it can, for instance, identify who must be informed in case of stabilization of new versions of the same object. Furthermore, CM can also inform object creators of most needed object characteristics in order to speed up the cooperation process. Finally, the cooperation manager must make sure that object versions which depend on other versions (e.g. **object-subobject** relationship) are not consolidated before the latter are.

4 Conclusions and Future Work

In this paper, we gave an overview of a design management and cooperation model and presented a mechanism to implement it. The model supposes a distributed computer system with a server-workstation architecture. It also assumes the existence of an underlying transaction manager that supports hierarchies of (long-duration) nested transactions. At least the top-most transactions should be allowed to exchange non-committed results.

The design management and cooperation model supports the specification and evolution of design projects. Object specification is done by means of application-oriented object

characteristics while design process specification is based on partial orders of design steps. The model also supports cooperation among designers who develop either common objects or sub-objects related to one another.

The design and cooperation mechanism which realizes the model relies on a so-called design and cooperation databse to control design process evolution. DBdc stores all relevant information about the design. The mechanism which is being implemented as two related software modules is responsible for DBdc's creation and maintenance. It also controls the correctness of the design process relying on partial orders of design steps and object-transfer operations.

A prototype of the design and cooperation mechanism is being implemented at the CPGCC-UFRGS. The DBdc and some functions of both the project and cooperation managers will be implemented as an application of the KRISYS (Knowledge Representation System). Other modules of the managers will be implemented in LISP and C. We expect to have a running version of the prototype by the end of 1992.

5 References

[1] P. A. Bernstein, V. Hadzilacos, N. Goodman: Concurrency Control and Recovery in Database Systems. Addison-Wesley Series in Computer Science, Addison-Wesley Publishing Co., 1987

[2] K. P. Eswaran et al. The Notions of Consistency and Predicate Locks in a Database System. Communications of the ACM, New York, v. 19, n. 11, p. 624-633, Nov. 1976

[3] T. Härder, A. Reuter: Principles of Transaction-Oriented Database Recovery. ACM Computing Surveys, v. 15, n. 4, p. 287-317, Dec. 1983

[4] C. Iochpe, M. A. C. Livi: Cooperation Support in a CAD Environment. 6th Brazilian Symposium on Database Systems, Manaus, May 1990 (in Portuguese)

[5] W. Käfer: A Framework for Version-Based Cooperation Control. Research Report, Universität Kaiserslautern, 1990.

[6] M. A. C. Livi: Requirements Analysis and the Proposal of a Cooperation Model for the AMPLO Environment. Research Report, CPGCC-UFRGS, 1991 (in Portuguese)

[7] N. M. Mattos: KRISYS - A Multi-Layered Prototype KBMS Supporting Knowledge Independence. Research Report, Universität Kaiserslautern, 1988

[8] M. H. Nodine, A. H. Skarra, S. B. Zdonik: Synchronization and Recovery in Cooperative Transactions. 4th Int. Workshop on Persistent Object Systems Design, Implementation, and Use, 1990

[9] F. de F. Rezende: A Transaction Model Supporting the World Concept of KRISYS. Master Dissertation, CPGCC-UFRGS, 1992 (in Portuguese)

[10] F. R. Wagner et al. STAR: An Environment for the Integration of Digital System Design Tools. 6th Congress of the Brazilian Microelectronics Society, Belo Horizonte, 1991 (in Portuguese)

Dynamic Multimodal Function Optimization using Genetic Algorithms

Walter Cedeño
Computing Research Group/CMRD
Lawrence Livermore National Laboratory
Livermore, CA 94550 (wcedeno@llnl.gov)
and
Venkateswararao Vemuri
Department of Applied Science and LLNL
University of California, Davis
Livermore, CA 94550 (vemuri@icdc.llnl.gov)

Abstract

Many optimization techniques work well for unimodal functions. If applied to multimodal functions, they tend to converge to only one of the many peaks. Optimization of multimodal functions becomes even more difficult if the function parameters change dynamically. Genetic algorithms have been successfully applied by several investigators for static optimization of multimodal functions. This modest success is primarily due to the ability of genetic algorithms to locate more than one peak. In this paper we introduce a combination of selection and replacement operators that is suitable for multimodal function optimization in a dynamic environment using various test functions, performance of this new operator is studied. Utility of this new operator to multimodal function optimization in a dynamic environment is described.

Keywords: genetic algorithms, multimodal functions, dynamic optimization.

1. Introduction

The problem of finding the maxima of functions with multiple peaks is difficult to solve using traditional optimization methods. Moreover, in some problems we want to find not only the highest peak, but all the other peaks as well. The problem becomes harder when the parameters of the function being maximized are changing dynamically. Problems of this type can be found in airline traffic control systems, satellite scheduling, and transportation scheduling and routing [1]. Good solutions are hard to find due to the problem complexity and good techniques, are most of the time, problem dependent. As the size of the problem becomes larger current techniques often produce solutions far from optimum at a very high computational expense. In this paper we describe a selection operator called *crowding selection* and a replacement operator applicable to dynamic multimodal function optimization. Our goal is to construct a GA model that is appropriate for this type of function optimization. Two cases are considered: static (the function parameters remain fixed), and dynamic (the parameters are allowed to vary).

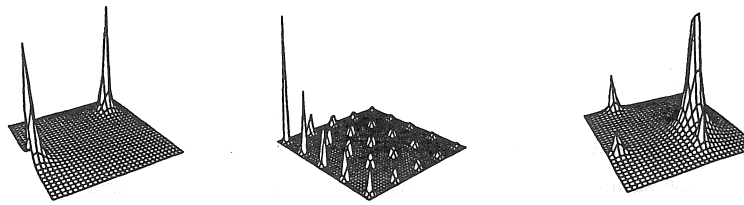
Genetic algorithms were introduced by Holland [9] in the 1970s, and have generated widespread interest in the last ten years. GAs are general purpose search procedures based on the principle of natural selection and genetic recombination. As in nature, GAs use the mechanics of evolution to improve a set of initial solutions using "recombination" and "mutation" of the "genetic material". This process ensures that good solutions are selected more often for recombination and generation of new solutions. During recombination new solutions are created and inserted into the current family of solutions; old and ineffective solutions are allowed to perish. Selection of mating partners, mating process, and replacement techniques are driven by genetic operators.

Genetic Algorithms differ from traditional techniques in several ways. GAs work with several solutions at the same time and improve them by recombining their good features while exploring new solutions in the search space. This approach has the advantage of permitting good solutions to guide the algorithm to near optimal solutions and at the same time prevents from getting stuck at local extrema. The genetic operators are governed by probability and are problem independent; the method can easily be applied to many problems. GAs use binary strings, to represent the problem, which are easy to manipulate thus facilitating fast execution of the programs.

Section 2 of this paper describes the set of test functions utilized to test different genetic operators, and benchmarks used to measure their performance. In section 3, we will introduce the GA model appropriate for multimodal function optimization. Section 4 presents the results obtained on the test functions and section 5 contains a summary and conclusions.

2. Function Definitions

To evaluate the performance of our GA model we use four different multimodal functions F1 to F4. The functions were chosen for their diversity: all peaks with approximately the same height, peaks with widely varying heights, and dynamically changing peaks. Figure 1 shows examples of these functions. Function F1 contains two peaks of the same height and width located far apart. Function F2 contains 25 peaks of different heights located in a 5x5 square region, the minimum height peak having been located in one corner and the maximum height peak in the opposite corner. This is one of the De Jong test function [4]. Due to its symmetry it is not sufficient to use this function to test the GA performance on multiple height peaks. Function F3 contains multiple peaks generated at random. The location, height, and width of each peak are all determined at random at runtime. Function F4, which is not shown, is made out of the sum of two different five-peak functions like F3; the contribution of each function is changed dynamically by multiplying it by a constant varying between 0.0 to 1.0. Different rates of contribution varying from 1% every generation to 1% every 10 generations were tried.



(a) (b) (c)

Figure 1: (a) Function F1 with similar peaks.
(b) Function F2 with multiple dissimilar peaks.
(c) Function F3 with randomly located peaks.

With the exception of F2, each peak in a function is generated by the formula:

$$F(x, y) = \frac{height}{1 + width * ((x - XCenter)^2 + (y - YCenter)^2)},$$

where *height* determines the maximum value for the peak, *width* the base width of the peak, and *XCenter* and *YCenter* the location of the peak in the x-y coordinate system. F2 is generated using the formula:

$$F(x, y) = 0.002 + \sum_{i=1}^{25} \frac{1.0}{i + (x - A_{1i})^6 + (y + A_{2i})^6},$$

where A_{1i} and A_{2i} respectively are the x and y coordinates of the i-th peak.

$$A = \begin{bmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & -16 & -16 & -16 & 0 & 0 & 0 & 0 & 0 & 16 & 16 & 16 & 16 & 16 & 32 & 32 & 32 & 32 & 32 \end{bmatrix}.$$

F2 [4] was chosen as a test function to compare the performance of our GA model in a search space where the taller peaks are situated near a corner. It also provides a complex search space where we can examine how the distribution of solutions are affected as the algorithm is run for longer periods.

3. Genetic Algorithms and Operators

Central to GAs are the twin concepts of natural selection and gene recombination. These algorithms manipulate strings of bits called *chromosomes,* which encode values in the search space. With this simple encoding, the genetic algorithm evolves a set of solutions, called a *population* , using the same mechanisms found in nature. They operate without any knowledge of the decoded search space. The only information they have is the *fitness* value (score) for each *organism* (solution) in the population and use it only to bias the selection of organisms for mating and/or replacement. Those organisms with higher fitness will tend to reproduce more and replace less fit organisms from the population.

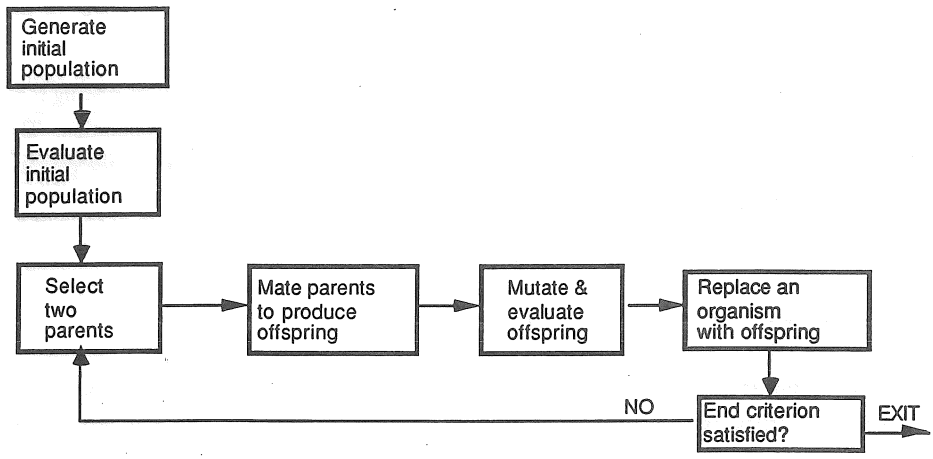


Figure 2: Basic structure of a genetic algorithm based on a steady-state model

Most of the material describing GAs uses the so-called *generational model*. In this model a new population replaces the current population entirely. This model lacks biological realism because organisms may survive various generations in a natural setting. In this paper we use what is known as a *steady-state* GA. In a steady-state GA, only one population evolves. A new organism is immediately inserted into the population using different replacement techniques. In Figure 2 we have the general structure of a steady-state GA. After generating the initial population the algorithms iteratively applies selection, mating, mutation, and replacement operators until the end criterion is satisfied or the maximum number of steps has been reached.

The *selection operator* determines which two organisms will mate. Normally those organisms with higher fitness values are selected more often for mating by using one of several techniques like roulette wheel selection, rank base selection, and so on. These approaches are not appropriate when the search space is multimodal, because it tends to bias selection towards one of the peaks. The approach we use here is named *crowding selection* . It is similar to *mating restriction* [5], where organisms from the same peak are selected to mate. The main difference is that in crowding selection one does not restrict mating to organisms associated with the same peak, but encourages similar organisms to mate. The first parent is selected sequentially from the population, thus allowing most of the organisms from the population to reproduce. The mating partner is selected as the most similar organism from a group formed by randomly selected organisms from the population. The similarity between two organisms is determined in the decoded search space using the Euclidean distance metric. This approach, combined with the replacement method described

below, allows organisms from different peaks to coexist while maintaining diversity in the population.

The *mating operator* simulates nature's DNA replication process. The goal of mating is to create new organisms (solutions) by combining features from organisms in the population. The simplest form of mating is crossover in which a division point is chosen at random and the parents' tails (bits to the right of the division point) are interchanged as shown below.

```
Parent 1: 1 0 0 1 0 1 1 1|1 0 0 0 0 1
Parent 2: 1 0 1 0 0 1 1 0|0 1 1 1 0 1
                          |-> crossover point chosen arbitrarily
Offspring 1: 1 0 0 1 0 1 1 1|0 1 1 1 0 1
Offspring 2: 1 0 1 0 0 1 1 0|1 0 0 0 0 1
```

After the offspring are generated the *mutation operator* is applied. It consists of iterating through the bits in each offspring and changing them from 1 to 0 or 0 to 1 with a very low probability. Mutation prevents the loss of characteristics from the population, and insures that the system evolves beyond local maxima.

The crux of the replacement method is to insert the offspring into the population using a *replacement operator*. It is this operator that ensures the survival of the fittest within the *species* (organisms from the same peak) by replacing the worst organism out of a group of most similar candidates. For each child, the operator forms *crowding factor groups* containing *crowding size organisms* selected at random from the population. The organism with the lowest fitness out of the most similar representative in each group is chosen to die. An example of this operator is shown in Figure 3. The main idea behind this operator is to increase competition among the members of the same species, allowing each species to evolve to it's best. The higher the crowding size the more probable the replaced organism is from the same species. The higher the crowding factor the better the chances are that the replaced organism has lower fitness. A similar technique [6] called *enhance crowding* has been used before, where the most similar organism out of a group of worst candidates is replaced. This technique is biased toward higher peaked organisms and tends to eliminate lower peaked organisms from the population rapidly, where as our technique keeps a majority of peaks for a longer period.

This GA model was found to be the best suited for the test functions described before. In the work done by others[4,5] the operators were either too restrictive or did not differentiate between elements in different peaks. For an operator to work correctly in a dynamic multimodal space, it must balance convergence in each peak with the search for other peaks. Crowding selection and the replacement method described above seem to achieve this balance.

Population	Fitness	Random Groups	Euclidean Distance	Similar Individuals
1011110101	15	1011110101	17	
0100010110	22	1011100111	13	0010010111
0100101110	18	0010010111	10	
1011100111	13	0100101110	8	
1101011110	6	0100010001	2	0100010001
0011010100	10	1111100110	20	
0100010001	5			
1111101010	9	1011100111	13	
0010010111	19	1101011110	17	1011100111
1111100110	14	1111101010	18	

Child to be inserted is 1011011110, replaces organism 0100010001.

Figure 3: Example of replacement operator with crowding size 3 and crowding factor 3.

4. Simulation Results

4.1 Selection of Metric

Before examining the results it is necessary to describe the metrics used to measure the performance in our GA model. The online and offline performance metrics as described in [4], were used to evaluate the convergence rate of our GA model for the static functions 1, 2, and 3. These values measure the population average and the best fitness average respectively over all generations. These online and offline metrics are not appropriate for dynamic functions because the best fitness and the average fitness of the function are not maintained. To evaluate the behavior of dynamic and static multimodal functions we introduce *organisms per peak* as a new metric to measure the number of organisms per peak after each generation. All organisms within certain distance from the location of the peak were associated with that peak. Organisms in overlapping regions were counted in both peaks and those not belonging in any peak are associated with another group. This metric allowed us to examine the competition between species from different peaks.

4.2 Genetic Algorithm Parameters

The GA parameters were set as follows:

Population size: 200 or more according to the number of peaks
Crossover probability: 0.95
Mutation probability: 0.06
Crowding selection group size: 4
Crowding Factor: 5
Crowding group size: 10
Number of chromosomes: 2
Bits per chromosomes: 30

These parameters were selected after various attempts and shown to work very well with the functions given above. For a particular function there might be a better combination of these parameters. Most of the time the GA was allowed to run for about 150 generations and then the results examined. In some cases, more generations were allowed in order to observe the effect of generations on the diversity of the population.

4.3 Results

The GA model performed very well for F1; each peak maintained similar number of organisms during all the generations. The peaks (see Figure 1c) have a height of 100, width of 0.0004, and located at (45000, 2000) and (15000, 62000). The coordinates of the search space were defined to be in the range (0, 65535) in both x and y directions. We observed an average of 66 organisms in peak 1 and 70 organisms in peak 2 with a variance of $\pm$ 11 from generation 5 to generation 150. Figure 4 contains the online and offline performance for F1. The offline and online performance for all the static functions were very similar. The performance in both cases increased almost monotonically in each generation. This occurs because the weak organisms in each peak are mostly replaced by better organisms and the best performer in each peak always survived. Analogous figures for F2 and F3 are not shown. Note that there is a period where the performance is low and then it increases rapidly until it stabilizes. The jump occurs when the GA finds very good representatives from the peaks. We also noted that even though the organisms are converging to the peaks, there were an average of 64 organisms not belonging to any of the peaks thus preserving some diversity in the population. This occurs because organisms from different peaks are allowed to mate, thus creating some organisms not belonging to any of the peaks.

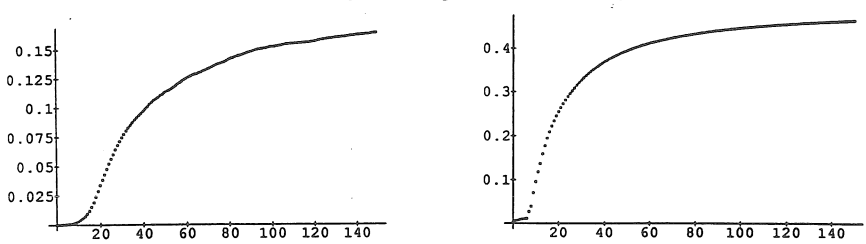


Figure 4: F1 a) online performance and b) offline performance

For the test functions with multiple height peaks, like F2 (Figure 1b) and F3 (Figure 1c), our GA model maintains a balanced distribution of the organisms between all peaks during the first 5-10 generations. After that those peaks with larger heights will start to dominate over the other peaks, but will not takeover the complete population. After many generations the number of organisms per peak will settle with shorter peaks having fewer organisms and higher peaks with more organisms. F3 has five peaks with the following configuration:

Peak no.	Location	Height	Width
1.	(140.586, 44.5314)	0.644506	0.0775816
2.	(102.585, 16.3897)	0.536561	0.76709
3.	(141.476, 5.18466)	0.39551	0.741381
4.	(153.506, 40.8626)	0.514533	0.799701
5.	(139.351, 43.8705)	0.889262	0.21302

Note that peaks 1 and 5 are very close that they practically merge to form the highest peak. Table 5 shows the organisms per peak for F3. Note also that the tallest peak, namely peak 5 has the maximum number of organisms after the initial has subside. Figure 6 shows snapshots of the location of the organisms for F2 at different generations. The location of most of the peaks can be seen from the graphs. Even after 100 generations a few organisms exist in the lower peaks.

Table 5: Organisms per peak for F3

Generation	Peak 1	Peak 2	Peak 3	Peak 4	Peak 5	Others
0	11	8	15	19	11	148
10	72	48	34	22	72	29
20	62	52	46	16	62	26
30	71	53	43	7	71	29
40	66	54	40	10	66	34
50	67	52	48	9	66	28
60	64	61	38	9	64	31
70	69	48	45	4	69	36
80	59	52	46	9	60	33
90	67	57	41	9	67	31
100	67	60	43	5	67	26
110	64	58	38	7	65	38
120	70	58	46	8	70	22
130	71	52	37	12	69	32
140	64	52	49	6	64	32
150	60	64	46	6	57	26

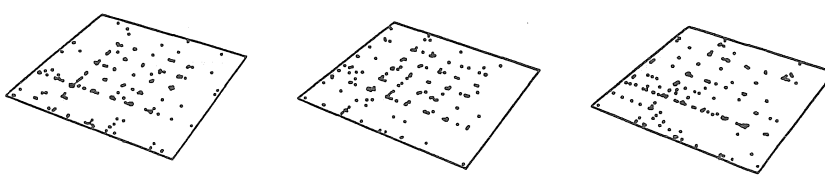


Figure 6: Location of organisms in F2 at generations 25, 50, and 100.

Results for F4 were encouraging as well. F4 was defined as a * F3 + b * G, where a varies from 0.0 to 1.0 and b from 1.0 to 0.0. F3 was defined above and G is a function similar to F3 with the following parameters:

Peak no.	Location	Height	Width
1.	(129.371, 10.0084)	0.907779	0.634976

2.	(109.307, 41.8726)	0.0708946	0.789199
3.	(127.675, 19.0206)	0.133323	0.886113
4.	(159.906, 58.8545)	0.806377	0.472369
5.	(139.827, 20.2571)	0.912816	0.166007

We ran our GA model with *a* and *b* changing 1% every generation and 1% every 10 generations. In both cases the GA was able to find the peaks of F3 during the first generations and then the peaks of G as the value of *b* kept increasing. Table 7 shows the organisms per peak for F4 changing 1% every generation. Note that the GA was able to maintain organisms in every peak most of the time.

Table 7: Organisms per peak for F4.

	Function F3					Function G					
Gen	Peak 1	Peak 2	Peak 3	Peak 4	Peak 5	Peak 1	Peak 2	Peak 3	Peak 4	Peak 5	Other
0	11	8	15	19	11	14	16	15	13	17	87
10	76	49	41	12	76	3	4	3	2	2	16
20	66	55	37	11	65	7	7	6	0	9	21
30	65	50	46	5	65	5	6	4	2	9	18
40	57	46	46	5	57	9	2	6	11	7	24
50	52	49	44	4	52	5	3	4	29	6	13
60	51	46	7	5	51	2	3	7	38	33	15
70	22	41	4	2	22	1	5	5	59	48	17
80	9	36	5	2	9	4	1	7	65	66	14
90	18	33	7	4	18	7	1	10	56	67	12
100	7	14	3	2	7	16	0	22	67	69	19

5. Comments and Conclusions

The GA model described above was shown to find solutions in both static and dynamic multimodal functions. For each case the GA model was able to identify the region where peaks are located and create a balanced environment between organisms from different peaks while at the same time converge to the top of peaks. The number of organisms in each peak was determined by the shape of the peaks. Competition between organisms in different peaks occurred when there was little room for improvement at the top of the peaks. Higher peaks will dominate in the population.

The results show our GA model as a viable tool for finding multiple solutions in a multimodal search space. We plan to applied our model to a combinatorial problem with multiple solutions to examine its behavior in such a complex space.

References

[1] L. Bodin, B. Golden, A. Assad, and M. Ball, "Routing and Scheduling of Vehicles and Crews, The State of The Art", *Computers and Operations Research* 10 (2), 63-211 1983.

[2] M. F. Bramlette, "Initialization, Mutation and Selection Methods in Genetic Algorithms for Function Optimization", *Proceedings of the Fourth International Conference on Genetic Algorithms*, R. K. Belew and L. B. Booker, eds., 100-107, Morgan Kaufmann Publishers San Mateo, CA, June 1991.

[3] L. Davis, (ed.), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold New York, NY, 1991.

[4] K. A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Doctoral dissertation, University of Michigan, Dissertation Abstracts International 36(10), 5140B, 1975.

[5] K. Deb and D. E. Goldberg, "An Investigation of Niche and Species Formation in Genetic Function Optimization", *Proceedings of the Third International Conference on Genetic Algorithms*, J. D. Schaffer, ed., 42-50, Morgan Kaufmann Publishers San Mateo, CA, June 1989.

[6] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading MA, 1989.

[7] D. E. Goldberg and J. Richardson, "Genetic Algorithms with Sharing for Multimodal Function Optimization", *Proceedings of the Second International Conference on Genetic Algorithms*, J. J. Grefenstette, ed., 41-49, Lawrence Erlbaum Associates, Hillsdale, NJ, June 1987.

[8] J. J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms", *IEEE Transactions on Systems, Man and Cybernetics*, SMC-16, no. 1, January-February 1986.

[9] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[10] J. D. Schaffer, "A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization", *Proceedings of the Third International Conference on Genetic Algorithms*, J. D. Schaffer, ed., 51-60, Morgan Kaufmann Publishers San Mateo, CA, June 1989.

Una Metodología Orientada por Objetos para el Diseño de Aplicaciones Gráficas Interactivas

Luis B. Chicaíza S.
Jorge A. Villalobos S.

Grupo de Investigación DFAC
Departamento de Ingeniería de Sistemas y Computación
Universidad de los Andes
e-mail: lchicaiz@andescol.bitnet
jvillalo@andescol.bitnet
Apartado Aéreo 4976 - Bogotá, Colombia

Resumen: Se propone un modelo conceptual y una metodología de diseño para aplicaciones gráficas interactivas. El propósito es mantener separados la interfaz y el control del resto de la aplicación, con el fin de facilitar su mantenimiento. Se propone el enfoque orientado por objetos como una manera natural de diseñar y especificar cada uno de los elementos del software gráfico. Las aplicaciones gráficas se caracterizan porque el usuario puede interactuar con las imágenes de los objetos del mundo en el cual ocurre el problema y las manipulaciones que él haga sobre cualquiera de sus imágenes deben generar manipulaciones sobre los objetos representados.

Descriptores: Programación Orientada por Objetos, Ingeniería de Software, Aplicaciones Gráficas Interactivas, Interfaces-Usuario

1. Introducción

La programación orientada por objetos (POO) se ha convertido en una de las herramientas que más expectativas genera en el ámbito de la ingeniería de software, debido, posiblemente, al potencial que encierra para aumentar la calidad de las aplicaciones y a su notable capacidad para estructurar el software con miras a facilitar su posterior mantenimiento, uno de los más grandes escollos en las metodologías tradicionales de desarrollo. En las aplicaciones gráficas interactivas se requieren extensiones a la metodología, en lo que respecta a la arquitectura del software obtenido, puesto que los elementos gráficos y de interacción no se acoplan de manera natural con el resto de los componentes de un programa.

Una *aplicación gráfica interactiva* (AGI) maneja una representación gráfica del estado del mundo del problema, siempre actualizada, y el usuario debe poder interactuar con esa imagen como si lo estuviera haciendo con los objetos del mundo que ésta representa. Además, puede haber más de una imagen asociada con un objeto del mundo, y se crean problemas de validez y coherencia entre ellas. Cada una de estas posibles imagenes se denomina una *vista*.

El objetivo de este artículo es mostrar un modelo conceptual para las AGI, con el fin de situar estructuralmente cada una de sus partes y definir sus relaciones, para proponer una metodología que permita abordar de manera sistemática el problema de diseñar este tipo de aplicaciones.

2. Arquitectura del Modelo

La idea fundamental de la POO es copiar la estructura y el comportamiento del mundo en el cual ocurre el problema, con el fin de obtener un modelo operacional de dicho mundo. Una vez obtenido este modelo, es posible estimularlo para obtener una reacción que se pueda interpretar como la solución del problema planteado. Bajo este esquema, la interfaz de una aplicación interactiva se puede ver como un administrador de ese modelo, que mantiene el diálogo con el usuario, y que va interpretando sus comandos para traducirlos en el conjunto de estímulos necesario para que el modelo responda y se llegue a un nuevo estado de la aplicación. Esto se ajusta bastante bien a las propuestas de separación entre la interfaz y el resto de la aplicación que, con el propósito de facilitar el mantenimiento, sugieren, entre otros, [OLS83], [SCH85] y [VIL87].

Una AGI está compuesta por un modelo operacional del mundo en el que ocurre el problema y uno o varios administradores de dicho modelo, que permiten a los diferentes tipos de usuario manejar vistas de los elementos del mundo. Cada administrador controla el diálogo con un usuario y mantiene una representación gráfica coherente con el estado del modelo.

2.1 Modelo detallado

Si se consideran las funciones del administrador, se puede llegar a un modelo más detallado, en el que se separen el control, la interacción y el manejo de la representación gráfica del mundo. A grandes rasgos, el modelo opera así: los eventos son tomados por los elementos de la interacción e interpretados; se envía dicha información al control de diálogo y éste se encarga de propagar sobre los cuatro mundos (incluido él) las consecuencias de esta orden del usuario. Esto quiere decir que el control es el encargado de mantener la coherencia y de controlar toda la aplicación.

La interacción con el usuario se va a reflejar en una extensión al modelo, para permitir la estimulación externa de los objetos mediante eventos. La separación de mundos y la caracterización de los objetos de cada mundo van a facilitar el planteamiento de una metodología de diseño.

2.2. El Modelo del Mundo del Problema

En este modelo están los objetos que existen en el mundo en el cual ocurre el problema: *los objetos reales*. No tienen imagen ni es posible interactuar directamente con ellos. Son una copia en estructura y comportamiento de los elementos del mundo real que intervienen en el problema.

2.3. El Modelo de la Interacción

La misión de este mundo es manejar la interacción con el usuario a través del mecanismo de eventos, generados a partir de un conjunto de dispositivos. En este mundo viven elementos tales como menús, ventanas, barras de scroll, botones, diales, etc. Se puede modelar como un conjunto de objetos que se encuentran a la espera de una interacción de parte del usuario. Cada uno de estos objetos se denomina un *objeto interactivo*. Dentro del mundo de la interacción existe también un manejador de eventos, que es capaz de generar un mensaje al objeto con el cual el usuario quiere interactuar. Este manejador de eventos es la única extensión al modelo de objetos.

Desde un punto de vista funcional, el manejador de eventos recibe las entradas generadas desde los dipositivos, identifica el objeto interactivo que debe activar y le envía a éste un mensaje con toda la información necesaria para que actúe. Un objeto interactivo es un elemento abstracto, sin imagen (vive en un mundo que no visualiza directamente el usuario), creado casi siempre desde el mundo del control para permitir la interacción con el usuario.

Un objeto interactivo sabe siempre responder al mensaje que envía el manejador de eventos para activarlo, con un método que decide cuál acción tomar, utilizando para eso su repertorio de eventos. Este método se denomina su *despachador*. El objeto puede realizar cuatro tipos de acciones al recibir el mensaje que lo activa: ignorar el evento, consumir la entrada (reaccionar modificando sus atributos), interpretar la entrada y enviar un mensaje a su padre con esa interpretación, o enviar

directamente la información del manejador de eventos a su padre. Los objetos interactivos también tienen métodos de creación (reciben a su padre como parámetro) y métodos para manejar su estado.

2.4. El Modelo de la Visualización

En el mundo gráfico viven las imágenes de los objetos reales y de los objetos interactivos. Aquí se encuentran desde la representación gráfica de un menú hasta las diferentes vistas de un objeto real. Cada una de estas imágenes se denomina un *objeto gráfico* y son el medio para que el usuario pueda visualizar el estado de los diferentes modelos que componen una AGI..

A diferencia de los objetos reales y de los objetos interactivos, los objetos gráficos no son independientes: siempre están relacionados con un objeto de otro mundo que se denomina su *objeto mellizo*. En el caso más general, tiene un mellizo en el mundo de la interacción y otro mellizo en el mundo real, dando lugar a un objeto real con una imagen con la cual es posible interactuar. Que un objeto gráfico no sea independiente significa que sus atributos gráficos son calculables a partir de los atributos de su objeto mellizo. Esto implica que debe existir una función que, a partir de un subconjunto de los atributos del objeto mellizo, es capaz de calcular los atributos gráficos. Esta función se llama la *función directa de relación.*

Con esta función es posible construir y manejar las imágenes de los objetos interactivos y una o varias vistas de los objetos reales, pero todavía no es suficiente para permitir la interacción directa con la imagen de un objeto real. Para esto, es necesario exigir que la función de relación sea biyectiva, de manera que sea posible propagar directamente la interacción del usuario, y se generen cambios de estado en el objeto mellizo correspondiente. Puesto que es biyectiva, existe su inversa, llamada la *función inversa de relación*, que permite calcular los atributos de los objetos mellizos a partir de los atributos gráficos.

2.5. El Modelo del Control

Este modelo representa la parte funcional de la aplicación; es responsable de coordinar el flujo interno de control y de garantizar la coherencia de los distintos modelos. Su comportamiento y estructura se deben diseñar a partir de los requerimientos del usuario y especificar mediante algún formalismo, para luego poder modelar este mundo en términos de objetos.

Un formalismo muy utilizado para especificar el mundo del control es el de los *diagramas de transición de estados* (DTE) [TOR87], [OLS83], en los cuales, utilizando un grafo, se muestra el flujo de control al interior de la aplicación, y se hace explícito el conjunto de servicios disponibles para el usuario en cada momento. Dado el caracter miltiventanas y multiusuario de las AGI, es necesario un conjunto de DTE para diseñar el control, con uno de ellos como inicial.

2.5.1. Objetos del Control

Al modelar el mundo descrito por un DTE a través de objetos, se obtiene un conjunto de elementos llamados *objetos de control* (cada uno representa un nivel), que van a manejar el flujo de la aplicación utilizando el mecanismo de mensajes. El objeto de control responde a los mensajes de sus hijos con un método que decide la forma de reaccionar, lo mismo que el objeto de control que debe crear antes de destruirse. Este método se llama el *despachador*. Existe, además, un método interno por cada procesador.

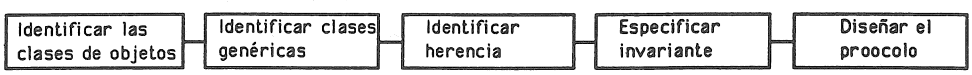
3. Una Metodología para Diseñar AGI

En esta parte se propone una metodología de diseño orientada por objetos que, partiendo de una fase previa de análisis, produce instancias válidas del modelo conceptual presentado anteriormente. Inicialmente se presenta una metodología general para diseño orientada por objetos, aplicable a cualquier mundo, y luego se harán especializaciones para ajustar la metodología a las características particulares de cada uno de los mundos de la aplicación.

3.1. Metodología de Diseño Orientada por Objetos

Una metodología no puede ser vista como un algoritmo infalible para obtener diseños óptimos, sino como una guía de apoyo para producir diseños de buena calidad. La metodología que se propone reconoce este hecho y ofrece un esquema ordenado para trabajar, dividido en una serie de etapas, y un conjunto de métodos asociado con cada una de ellas.

Se propone una metodología de cinco pasos:



3.1.1. Identificar las Clases de Objetos

En esta fase se identifican todas las clases de objetos necesarias para modelar el mundo del problema. Se parte del conocimiento obtenido en la etapa de análisis y se llega a una estructura del modelo, denominada el *grafo de niveles de composición* (GNC).

Los atributos de una clase son la información necesaria (objetos de otra clase) para abstraer y modelar el estado de un tipo de elemento del mundo. Se deben agregar atributos a una clase hasta que todas las características interesantes en el mundo del problema estén incluidas [MEY88]. Esta estructura da lugar a un grafo dirigido (el GNC del modelo) cuyos vértices son las clases y los arcos la relación de pertenencia (clase → atributo). Las clases de primer nivel son aquellas que no

tienen predecesores (clases externas) y las de último nivel las que no tienen sucesores (clases de base del sistema).

Se proponen dos formas de identificar las clases de objetos y crear el GNC. Una, utilizando el enfoque de arriba hacia abajo [JAL89], y la otra avanzando en el proceso de identificación de abajo hacia arriba [MEY88], aunque las combinaciones de las dos resultan muy adecuadas.

Método de arriba hacia abajo: 1) identificar los objetos de primer nivel tomando como guía aquellos que están directamente involucrados en el problema y que se pueden considerar como los más "grandes" en el mundo , 2) para cada una de las clases identificadas modelar su estado a través de un conjunto de atributos y adicionar las clases de estos al grafo de niveles de composición, 3) repetir el segundo paso hasta llegar a las clases de último nivel.

Método de abajo hacia arriba: 1) partiendo de las clases de último nivel construir el siguiente nivel de composición, e identificar las clases que se pueden modelar con atributos que ya están en el grafo, 2) repetir el paso uno hasta llegar a los objetos de primer nivel.

3.1.2. Identificar las Clases Genéricas

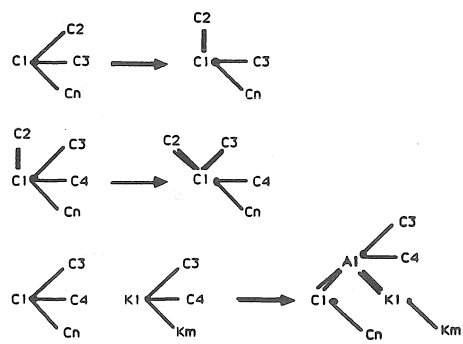
La entrada a esta etapa es el GNC del modelo y la salida es una nueva versión del GNC, con algunas de las clases parametrizadas, y un conjunto de esquemas de clase que las soportan.

3.1.3. Identificar las Relaciones de Herencia

La entrada a esta etapa es el GNC que se obtiene de la etapa anterior y la salida es el GNC aumentado con las relaciones de herencia (GNCH). Se coloca un arco de herencia entre dos clases del GNC (clase1 → clase2) si la clase2 hereda de la clase1.

Método de análisis sintáctico:

- buscar en el GNC alguno de los patrones del lado izquierdo de las reglas de reescritura:



- evaluar si en el mundo la relación de herencia que sugiere la regla tiene sentido, es decir, que sus comportamientos coinciden (análisis semántico)
- aplicar la regla de reescritura sobre el GNC para identificar la herencia simple (regla 1), la herencia múltiple (regla 2) y las clases abstractas (regla 3)

3.1.4. Especificar el Invariante de las Clases

El *invariante* de una clase es un predicado sobre el estado de un objeto que dice si éste es válido en el modelo. Se define así un subespacio sobre el espacio de todos los estados posibles de un objeto, en el que únicamente existen elementos que modelan objetos válidos en el mundo del problema. Se propone el siguiente método para construir el invariante de cada clase: 1) recorrer de arriba hacia abajo las jerarquías de herencia y escribir el invariante de cada clase, 2) para cada clase utilizar el GNC para identificar sus atributos y colocar una aserción que relacione los valores de los atributos y defina condiciones de validez de cada uno de ellos, 3) si la clase hereda de otras, construir el invariante a partir de la conjunción de los invariantes de las clases de las cuales hereda, y agregar una aserción que imponga nuevas restricciones y/o relaciones entre los atributos nuevos.

3.1.5. Diseñar el Protocolo de las Clases

Al exterior del modelo los únicos métodos visibles deben ser aquellos de las clases de primer nivel. No es posible utilizar los servicios de los atributos directamente porque, en tal caso, se podría perder la coherencia: sería posible alcanzar un estado en el cual el atributo cumpliera el invariante de su clase, pero el objeto que lo posee no cumpliera con el suyo. Esta etapa se divide en tres partes: identificar las operaciones de cada clase, verificar la validez y la completitud del diseño, especificar cada método. Para identificar las operaciones de cada clase se proponen tres métodos:

Método basado en la teoría de Tipos Abstractos de Datos: Ver [CAR91].

Método de Propagación de Necesidades: 1) identificar los servicios que debe proveer el modelo hacia el exterior y asociarlos con operaciones de las clases de primer nivel, 2) propagar dentro del GNC las necesidades que se van generando de una clase hacia sus atributos

Método de Propagación de Servicios: 1) especificar las operaciones de los objetos de último nivel, pensando en los servicios que este tipo de elementos debe prestar en el mundo, 2) propagar hacia arriba los servicios que ofrecen las clase de bajo nivel, teniendo en cuenta los servicios que ofrecen los atributos para diseñar las operaciones de cada clase

Para verificar la validez y completitud del diseño se debe tener en cuenta: 1) cada método de cada clase debe ser expresable en términos de los métodos de sus atributos, 2) los métodos de las clases

del ultimo nivel deben ofrecer todos los servicios externos que se requieren desde otros modelos. Para especificar un método se construye su precondición y su postcondición.

3.2. Diseño Orientado por Objetos de AGIs

En esta parte se especializa la metodología planteada en la sección anterior, para ajustarla al diseño de cada uno de los mundos que conforman una AGI.

3.2.1. Diseño del Modelo del Mundo

El método que se propone corresponde exactamente a la metodología general presentada; la única precisión que se debe hacer es en el diseño del protocolo, en el cual, los servicios externos al modelo se deben definir a partir de los requerimientos establecidos en el modelo de control, haciendo necesario avanzar en este segundo modelo para poder evaluar completitud

3.2.2. Diseño de la Interfaz

Se utiliza la metodología general propuesta, con las siguientes precisiones:

- utilizar los recursos (librerías, etc.) que ofrece el sistema como clases de último nivel al crear el GNC del modelo
- sólo clases interactivas y tipos simples pueden pertenecer al GNC del mundo de la interfaz
- para cada objeto interactivo: 1) determinar y codificar los eventos a los cuales debe responder, 2) la forma de reaccionar y 3) asociar un método con cada evento, teniendo en cuenta que para los objetos interactivos compuestos deben incluirse los eventos de sus componentes
- las constructoras deben recibir como argumento el objeto que lo crea.
- El protocolo consta de métodos contructores, de reacción a eventos (uno por cada evento), y el despachador.
- La especificación de un método de reacción a eventos, incluye en la postcondición el tipo de acción que toma como respuesta al evento que lo activa. La manera de notificar a su padre es llamar su método despachador.

3.2.3. Diseño del Control

La entrada a esta etapa es el conjunto de categorías de usuarios, el conjunto de requerimientos y el bosquejo de la interacción. Se propone una metodología de dos etapas:

- construir el conjunto de DTE
- traducir los DTE a objetos

3.2.4. Diseño de la Representación Gráfica

El propósito de esta etapa es determinar la manera como se van a visualizar los objetos de la AGI. Las entradas son el bosquejo de la interacción que desea el usuario, la información sobre las vistas,

el conjunto de DTE y los modelos del mundo real y de interacción. Se propone un método que procesa todos los elementos del conjunto de DTE en los siguientes pasos:

- determinar el conjunto de vistas
- diseñar los objetos gráficos
- establecer las funciones de relación
- enriquecer los modelos real e interactivo.

4. Conclusiones

En este artículo se presenta una propuesta de metodología de diseño orientada por objetos para aplicaciones gráficas interactivas. Más que una solución al problema que plantea el desarrollo de este tipo de aplicaciones, se pretende establecer una metodología, con la cual sea posible enfrentar este problema de una manera sistemática. La metodología que se propone en el artículo ha sido probada en el desarrollo de software gráfico para programación de robots fuera de línea, que incluye todas las características de una AGI. En este caso se utilizó C++ como lenguaje de implantación.

6. Bibliografía

[CAR91] Cardoso, R., Verificación y Desarrollo de Programas, Ediciones Uniandes, agosto 1991.

[JAL89] Jalote, P., Functional Refinement and Nested Objects for Object-Oriented Design, IEEE Transactions on Software Engineering, vol. 15, No. 3, Marzo/89.

[MEY88] Meyer, B., Object-Oriented Software Construction, Prentice-Hall, 1988.

[OLS83] Olsen, D., Dempsey, E., SYNGRAPH: A Graphical User Interface Generator, Computer Graphics, Vol. 17, No. 3, 1983.

[SCH85] Schulert, A., Rogers, G., ADM - A Dialog Manager, CHI'85 Proceedings, 1985.

[TOR87] Toro, V., Diseño, Especificación y Prototipificación de Sistemas Interactivos, XIII Conferencia Latinoamericana de Informática, 1987.

[VIL87] Villalobos, J., Arquitectura de una Interfaz-Usuario: Un Modelo General, XIII Conferencia Latinoamericana de Informática, 1987.

[BUX83] Buxton, W., Lamb, M., Towards a Comprehensive User Interface Management System, Computer Graphics, Vol. 17, No. 3, 1983.

UN ALGORITMO DISTRIBUIDO DE DETECCION DEL INTERBLOQUEO PARA EL MODELO UNICO-RECURSO

Alberto Córdoba,
Informática Ercilla
Rodriguez Arias, 23
48011 Bilbao, España

José R. González de Mendívil,
Dept. Electricidad y Electrónica
Universidad del País Vasco
48080 Bilbao, España

José R. Garitagoitia
Dept. Informática
Universidad del País Vasco
48080 Bilbao, España

RESUMEN: En este artículo se presenta un algoritmo distribuido de detección del interbloqueo para un Sistema Distribuido de Base de Datos con modelo de ocupación único-recurso. El algoritmo de detección propuesto sigue la filosofía de los algoritmos arco a cazar y presenta las siguientes ventajas: (a) Solo utiliza información local del Grafo de Esperas; (b) todos los mensajes utilizados en el proceso de detección tienen longitud constante y mínima información; (c) el algoritmo se demuestra formalmente correcto y no genera falsos interbloqueos; (d) por cada interbloqueo sólo un único agente de transacción es responsable de la resolución; (e) el número de mensajes intercambiados para detectar el interbloqueo es menor que el de otros algoritmos de características similares.

PALABRAS-CLAVE: Sistemas Distribuidos; Sistemas de Bases de Datos; Modelo de Ocupación Unico-Recurso; Detección del Interbloqueo; Detección Distribuida.

I.- INTRODUCCION

El *Interbloqueo* (*Deadlock*) es un problema muy importante en los Sistemas Distribuidos de Bases de Datos. Las tres técnicas básicas para abordar dicho problema son [1]: (i) *Prevención*; (ii) *Exclusión*; (iii) *Detección* y *Resolución*. Reconociendo que en los Sistemas Distribuidos de Bases de Datos, las técnicas de Prevención y Exclusión no pueden ser aplicadas [2] [3], centraremos el problema en las técnicas de detección.

Los mecanismos de control de concurrencia usuales, garantizan la consistencia de las operaciones sobre los recursos de la Base de Datos, pero fuerzan a las transacciones a esperar si los recursos que solicitan no se encuentran disponibles [4]. Si estas esperas no son controladas se puede producir una situación en la que un conjunto de transacciones esperan indefinidamente por los recursos asignados a transacciones en el mismo conjunto. Esta situación no deseable es la que se denomina *situación de interbloqueo* [5]. Las técnicas de detección deben realizar dos importantes tareas [3]: (1) mantener actualizada la información de las interacciones entre las transacciones; y (2) buscar sobre dicha información grupos de transacciones en espera circular. Cuando una situación de interbloqueo es detectada se requiere un *mecanismo de resolución* que permita a ciertas transacciones en el interbloqueo continuar su ejecución [1] [6]. Los algoritmos de detección para Sistemas Distribuidos se clasifican según la forma en que se realicen las tareas (1) y (2) del proceso de detección [2] [3]: (i)*Centralizados*. Un único lugar del sistema se encarga de mantener actualizada la información y de realizar las búsquedas de transacciones en espera circular. (ii) *Jerárquicos*. Los lugares se encuentran organizados de forma jerárquica y se detecta el interbloqueo en los lugares inferiores. (iii) *Distribuidos*. Todos los lugares del sistema participan de la misma manera en el proceso de detección. Las ventajas e inconvenientes de estos tipos de algoritmos se pueden estudiar en [2]

[3] [7]. Además en el caso de las Bases de Datos Distribuidas, la complejidad de los algoritmos de detección depende del tipo de modelo de ocupación de recursos que la Base de Datos permite [2].

En este artículo presentamos un algoritmo distribuido de detección y resolución del interbloqueo para un Sistema Distribuido de Base de Datos con un modelo de ocupación único-recurso. El modelo de la Base de Datos sobre el cual se formalizan las acciones del algoritmo es un modelo basado en Grafos debido a Menasce y Muntz [8]. El algoritmo es del tipo *arco a cazar* (*edge-chasing*) y está basado en la idea original de Mitchell y Merritt de asociar a cada agente de transacción dos etiquetas [9]. Mediante la introducción de un nuevo mecanismo que permite distinguir la formación secuencial del interbloqueo de la formación simultánea se reduce el número de mensajes necesarios para la detección. Este coste es variable e igual a n(n+2)/4 en un peor caso de formación del interbloqueo y n en el mejor caso (n- número de lugares en el interbloqueo).

El resto del artículo se organiza de la siguiente forma: En la Sección II se muestra el modelo de Base de Datos utilizado y se caracteriza la situación de interbloqueo. En la Sección III se introduce el algoritmo de detección propuesto. En la Sección IV se muestra un ejemplo de funcionamiento del algoritmo, así como el costo en número de mensajes necesarios para la detección. Por último, la Sección V se dedica a las conclusiones del trabajo.

II.- UN MODELO DE BASE DE DATOS

Un Sistema Distribuido de Base de Datos se puede considerar como un conjunto de lugares que se encuentran conectados a través de una red de comunicaciones intercambiando información mediante el envío y la recepción de mensajes [6]. Cada lugar tiene un único identificador, asumiendo que este identificador es un número natural, el conjunto de lugares se define como $\mathbf{L}= \{L_j / L_j \in \mathcal{N}, j:1..N\}$. La red de comunicaciones se supone perfectamente fiel: los mensajes enviados a través de la red no se pierden, se reciben en un tiempo finito y están libres de error. Cada lugar L_j se puede considerar como una Base de Datos central que gestiona una porción de la Base de Datos global. Los recursos almacenados en L_j sólo pueden ser accedidos por medio del controlador de recursos del lugar C_j.

Los usuarios interaccionan con los recursos de la Base de Datos por medio de la ejecución de transacciones [6]. Una transacción se puede considerar, de forma simplificada, como un conjunto de operaciones de *lectura* y *escritura* sobre los recursos de la Base de Datos. Cada transacción tiene un único identificador que la distingue del resto de transacciones en el sistema. Asumiendo que este identificador es un número natural, el conjunto de transacciones en el sistema se define como $\mathbf{T}= \{T_i / T_i \in \mathcal{N}, i:1..M\}$. Una transacción T_i es local en L_j si actúa sólo sobre los recursos gestionados por C_j. Una transacción es distribuida si solicita recursos a distintos controladores. Para realizar las operaciones sobre los distintos controladores, cada transacción sea local o distribuida tiene un único representante en cada lugar denominado *agente de transacción* [6] [8]. Un agente de transacción, denotado de la forma $a_{ij}= (T_i, L_j)$ es un proceso que actúa en cada lugar en nombre de la transacción.

Para representar el estado de ejecución de todos los agentes de transacción se utiliza el modelo denominado Grafo de Espera entre Transacciones [8]. Este grafo, denotado **G**, es un

grafo dirigido y dinámico que está formado por una pareja (X, U), donde X es el conjunto de nodos y U es el conjunto de arcos. El conjunto de nodos X está formado por todos los agentes de transacción, de forma que X= TxL. El conjunto de arcos verifica en todo instante U⊂ XxX. Sólo son posibles dos tipos de arcos en U: (i) *Arcos internos*. Son arcos de la forma u= (a_{ij}, a_{kj}), establecidos en un mismo lugar L_j entre dos agentes de transacciones distintas. La existencia de un arco de este tipo indica que el agente a_{ij} espera por algún recurso asignado al agente a_{kj}. (ii) *Arcos externos*. Son arcos de la forma u= (a_{ij}, a_{ik}), establecidos entre los agentes de una misma transacción en lugares diferentes. En un arco externo u= (a_{ij}, a_{ik}), el agente a_{ij} se denomina *saliente* del lugar L_j y el agente a_{ik} se denomina *entrante* en el lugar L_k.

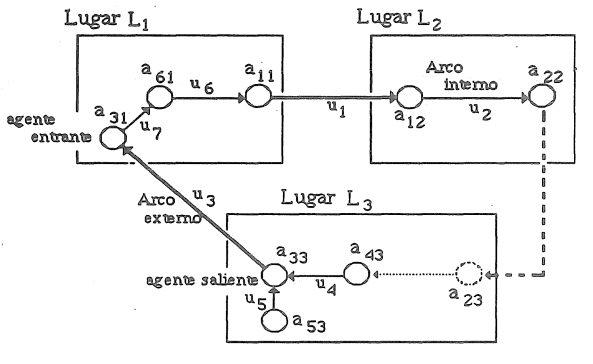


Figura 1Grafo de Espera entre Transacciones. Si se crea el arco externo (a22, a23) y el agente a 23 se queda a la espera del agente a 43 entonces se produce un circuito en el grafo y por lo tanto una situación de intrebloqueo que envuelve a los tres lugares.

En el modelo de ocupación único-recurso, un agente sólo puede adquirir un único recurso cada vez. Por lo tanto, un agente sólo puede esperar por otro agente. Esto queda caracterizado sobre el grafo G por el hecho de que para cada nodo a∈ X se verifica estrictamente la relación $0 \leq d^+(a) \leq 1$, donde $d^+(.)$ representa el semigrado positivo de un nodo [10].

La condición necesaria y suficiente para que exista interbloqueo en el sistema es que exista un circuito de nodos en el grafo G [8] (**figura 1**). Un algoritmo de detección se encarga básicamente de encontrar circuitos sobre el grafo.

III.- EL ALGORITMO DE DETECCION

El algoritmo de detección propuesto se clasifica en la categoría de los algoritmos arco a cazar [2]. Estos algoritmos basan su funcionamiento en el envío de mensajes especiales de detección denominados *sondas*, a través de los canales representados por los arcos externos en el grafo. No existe ningún lugar en el que se centraliza la información del grafo. Todos los lugares mantienen actualizada la información de aquella parte del grafo correspondiente únicamente al lugar. Cada lugar tiene una copia del algoritmo de detección y todos los lugares cooperan equitativamente en el proceso de detección enviando las correspondientes sondas en la dirección contraria a la dirección de los arcos en el grafo. Se supone que las sondas de un lugar se reciben en el mismo orden en que fueron enviadas, en un tiempo finito y libres de error. Se supone que no existen suspensiones espontáneas de los agentes de transacción con objeto de evitar la detección de falsos interbloqueos [3].

Cada nodo establecido en un lugar tiene asignados los siguientes elementos representativos: (i) Identificador privado (*identificador*). Es aquel número natural que se corresponde con el identificador de la transacción que el nodo representa en el lugar. Este número es fijo y no

puede ser modificado. (ii) Identificador público (*valor*). Es un número entero positivo que puede modificarse por el algoritmo de detección. (iii) *Marca*. Es un valor booleano asociado al nodo que puede modificarse por el algoritmo de detección.

En todo instante de tiempo se marcan aquellos nodos que son el final de un camino abierto de esperas. Cuando un nodo marcado se convierte en un nodo saliente, su correspondiente nodo entrante en otro lugar se marcará, generando sondas especiales con marca si el nodo entrante se queda a la espera directa o transitiva de otro nodo que también es saliente. En caso contrario, el nodo entrante marcado cederá la marca a los nodos que espera, volviendo a establecer la marca en los nodos que son el final de un camino abierto de esperas. Las sondas marcadas generadas en este proceso no se pierden porque sus valores quedan almacenados en los valores de los nodos salientes. Los valores de las sondas marcadas siempre se comparan con los valores asociados a los nodos marcados. Las posibles sondas que se produzcan cuando no existe ningún nodo marcado, estarán sin marcar y sus valores, en este caso identificadores, se compararán con los identificadores de los nodos no marcados. En la **figura 2** se muestran las definiciones de las principales funciones utilizadas por el algoritmo de detección.

L= {j / j:1..N} conjunto de lugares del sistema.
T= {i / i:1..M} conjunto de identificadores.
G= (**X**, **U**) Grafo de Espera entre Transacciones
V= {v / v∈ Z, v ≥ 0} conjunto de valores

Funciones:

Identificador. id: **X** ---> **T**, ∀aij ∈ **X**, id(aij)= i.
Valor. val: **X** ---> **V**, ∀aij ∈ **X**, val(aij)= v.
Marca. marca: **X** ---> {true,false}, ∀aij ∈ **X**, marca(aij)
Sucesor local. akj es sucesor local de aij, akj suc aij, sii:
(i) ∃ u∈**U** / u=(aij, akj) ó
(ii) ∃ amj ∈ **X** y u'∈ **U** / u'=(aij, amj) y akj suc amj
Antecesor local. aij es antecesor local de akj, aij ant akj sii:
(i) ∃ u∈**U** / u=(aij, akj) ó
(ii) ∃ amj ∈ **X** y u'∈ **U** / u'=(aij, amj) y akj ant amj.
Entrante. entrante: **X** ---> {true,false}, entrante(aij)= true sii:
∃ u∈**U** / u=(aik, aij) con k≠ j.
Saliente. saliente: **X** ---> {true,false}, saliente(aij)= true sii:
∃ u∈**U** / u=(aij, aik) con j≠ k.

Figura 2. Definiciones utilizadas por el algoritmo de detección.

Las sondas que se utilizan para detectar el interbloqueo contienen la siguiente información s = (marca, id_s, receptor). El primer campo booleano indica si la sonda es una sonda marcada o no. El identificador de la sonda id_s es un valor adquirido del valor asociado a un nodo o bien un identificador de un nodo. Por último *receptor* es el nodo al cual va dirigida la sonda. Dado que una sonda se transmite a través de un arco externo $(a_{ij}, a_{ik}) \in U$ en la dirección opuesta al arco y dado que una sonda se genera por un nodo entrante a_{ik} y el receptor es el correspondiente nodo saliente a_{ij}, se puede obviar la actualización del campo receptor en la sonda en la presentación del algoritmo por criterios de simplicidad. Por otra parte, asumiremos que s* indica una sonda marcada, siendo s*= id_s; y s indica una sonda sin marcar, siendo s= id_s (id_s = identificador o valor).

En el modelo único recurso todos los circuitos que se pueden formar son disjuntos es decir no tienen arcos comunes. Por lo tanto se puede plantear un esquema de detección en cada lugar que trate sobre los circuitos locales y un mecanismo de detección independiente para circuitos implicado distintos lugares. Vamos a definir un circuito [10] implicando n lugares por las relaciones de los arcos externos que contiene o de forma equivalente por la secuencia de nodos entrante y saliente que forman parte del circuito en cada lugar.Sea $[a_{ij}, a_{kj}]$ una relación entre dos nodos $a_{ij}, a_{kj} \in X$, tal que verifica: (i) entrante(a_{ij})= true; (ii) saliente(a_{kj})= true; (iii) a_{kj} suc a_{ij}. Se define un circuito $\mathscr{C}$ implicando n lugares de la forma:

$\mathscr{C} \equiv [a_{11}, a_{21}][a_{22}, a_{32}][a_{33}, a_{43}]... [a_{ii}, a_{(i+1)i}] ... [a_{(n-1)(n-1)}, a_{n(n-1)}][a_{nn}, a_{1n}]$

donde se han utilizado índices correlativos de lugares y de identificadores de transacción, pero en este caso y para no perder generalidad los identificadores de transacción deben interpretarse en general como $T_i \in T$. Es muy sencillo demostrar que la definición anterior de circuito corresponde realmente a un circuito en el grafo G.

El proceso de detección completo realiza dos tareas: (1) actualización del grafo G y (2) búsqueda de circuitos implicando varios lugares. La tarea (1) también es en parte gestionada por los mecanismos de control de la transacciones en lo que respecta a la creación y desaparición de agentes de transacción (nodos del grafo). El algoritmo de detección completo comprende tres pasos denominados: (i) Paso de actualización de valores y marcas (**figura 3**); (ii) Paso de generación de sondas y (iii) Paso de emisión de sondas y detección (**figura 4**).

A1: $\exists$ u= (aij, akj) en un instante t sii el nodo aij espera por akj por algún recurso.
A2: $\exists$ u= (aij, aik) en un instante t sii el nodo aij espera por algún mensaje de su agente aik en el lugar k.
A3: En el instante en que aparece el arco externo u= (aij, aik) saliente(aij)= true, val(aij)= 0, entrante(aik)= true, val(aik)=id(aik).
A4: Si un nodo aij cumple entrante(aij)= false y d+(aij)=0 entonces val(aij)=0.
A5: Sea un nodo aij tal que saliente(aij)=true y $\forall$ akj / akj ant aij se cumple entrante(akj)=false, entonces si aij recibe s* y se cumple s* > val(aij) el nuevo valor de aij es val(aij)= s*.
A6: Si en un instante se cumple entrante(aij)= true, marca(aij)= false y se crea un arco por **A1** entonces $\forall$ akj / akj suc aij:
if d+(akj)= 0 then marca(akj)= true;
if saliente(akj)= true then para u=(akj, akp)
    if d+(akp)= 0 then marca(akp)= true.
A7: Sea akj / entrante(akj)= false, saliente(akj)= false, marca(akj)= true por **A6** entonces $\forall$ amj / amj suc akj:
if d+(amj)= 0 then marca(amj)= true, marca(akj)= false;
if d+(amj)= 1 and saliente(amj)= true then $\forall$ aij / entrante(aij)= true y verifica aij ant akj: marca(aij)= true, marca(akj)= false.
A8: Si en un instante se cumple para un nodo entrante(aij)= true y marca(aij)= true entonces:
$\forall$ akj / akj suc aij: if d+(akj)= 0 then marca(akj)=true, marca(aij)=false.
A9: Si para un nodo se cumple marca(aij)=true por **A5**, **A6** o **A7** y por **A2** se crea un arco externo u=(aij, aik) entonces marca(aij)= false, mrca(aik)= true.

Figura 3. Paso de Actualización de valores y marcas

Las reglas de la **figura 3** se utilizan para distinguir la formación secuencial de la formación simultanea de parejas de relaciones $[a_{ij}, a_{kj}]$, además por las reglas A6, A7, A8 y A9 cuando se crea una relación $[a_{ij}, a_{kj}]$ entonces sólo puede verificarse que marca(a_{ij})= true o false y que siempre marca(a_{kj})= false. La regla A5 asegura que los valores de las sondas marcadas s* nunca se pierden y son almacenados en los valores de los nodos salientes. El paso de generación de sondas y de emisión de sondas (**figura 4**) sólo funciona a través de las relaciones $[a_{ij}, a_{kj}]$ establecidas en un lugar. Ciertas actualizaciones de valores distintas a A3, A4 y A5 se realizan en los pasos de generación y de emisión de sondas. Una especificación formal del algoritmo completo basado en la Teoría de Autómatas se encuentra en [11] y bajo esta formulación se demuestra la prueba formal de corrección. El algoritmo asegura que por cada circuito sólo un nodo detecta el interbloqueo y por lo tanto dicho nodo puede ser víctima con objeto de resolver de manera eficaz el interbloqueo.

IV.- FUNCIONAMIENTO DEL ALGORITMO

En esta Sección mostramos a través de un ejemplo el funcionamiento del algoritmo de detección propuesto. Así mismo, se presentan los costos en número de mensajes necesarios para la detección. Supongamos que en cada lugar L_1, L_2, L_3 y L_4 (**figura 5**) existe inicialmente un agente activo, denominados a_{11}, a_{22}, a_{33}, a_{44} respectivamente, con id(a_{11})=1, id(a_{22})=2, id(a_{33})=3, id(a_{44})=4. Supondremos el instante t_i anterior al t_{i+1}.

En el instante t1 se crea el arco externo (a_{11},a_{12}) por A2, y el arco interno (a_{12},a_{22}) por A1. Por A3 val(a_{11})= 0 y val(a_{12})= id(a_{12})= 1, y por A6 marca(a_{22})= true.

Paso de Generación de sondas: Se activa en un lugar en un instante de tiempo t cuando aparece una relación [aij, akj]:

Paso 1.1. Generación de sondas marcadas

```
if marca(aij) then
  if val(aij)> val(akj) then
    begin s*:= val(aij); emitir(s*) end
  else
    begin val(aij):= val(akj); s*:= val(aij); emitir(s*) end
else
  if val(akj)>0 then begin s*:= val(akj); emitir(s*) end;
```

Paso 1.2. Generación de sondas sin marcar

```
if not marca(aij) and v(akj)= 0 then
  if id(aij)> id(akj) then
    begin s:= id(aij); emitir(s) end;
```

Paso de emisión de sondas y detección: Se activa en un lugar en un instante de tiempo t cuando para una relación existente [aij, akj] el nodo akj es receptor de una sonda:

Paso 2.1. Sondas marcadas (se recibe s*)

```
if marca(aij) then
  begin
    if s*= val(aij) then victima(aij); { Interbloqueo}
    if s*> val(aij) then
      begin val(aij):= s*; val(akj):= s*; emitir(s*) end
  end
else
  if s*> val(akj) then
    begin val(akj):= s*; emitir(s*) end;
```

Paso 2.2. Sondas sin marcar (se recibe s)

```
if not marca(aij) and val(akj)= 0 then
  begin
    if s= id(aij) then victima(aij);{ Interbloqueo}
    if s> id(aij) then emitir(s)
  end;
```

Figura 4. Paso de Generación de sondas, Paso de Emisión de sondas y detección .

En el instante t2 se crean simultaneamente los arcos externos (a_{22},a_{23}), (a_{33},a_{34}), (a_{44},a_{41}) por A2, y posteriormente los arcos internos (a_{23},a_{33}), (a_{34},a_{44}), (a_{41},a_{11}). Por A3 val(a_{22})= 0, val(a_{23})= id(a_{23})= 2, val(a_{33})= 0, val(a_{34})= id(a_{34})= 3, val(a_{44})= 0, val(a_{41})= id(a_{41})= 4. Además por A9, marca(a_{22})= false y marca(a_{23})= true. Esta situación de interbloqueo es la reflejada en la **figura 5**.

En cada lugar aparece una relación [entrante,saliente] que activa el paso de generación de sondas. En L1 se genera una sonda s1 por el Paso 1.2, tal que s1= id(a_{41})= 4, que se envía a L4. En L2 por el Paso 1.2 no se genera sonda. En L3 se genera una sonda s3* por el Paso 1.1, tal que s3*= id(a_{23})= 2, que se envia a L2. En L4 por el Paso 1.2 no se genera sonda.

La sonda s1 emitida por L1 al llegar a L4 activa en ese lugar el Paso 2.2 de emisión y detección de sondas. Por este Paso 2.2 se genera una sonda s'1= s1 que se emite hacia L3. cuando s'1 llega a L3 activa el Paso 2.2 y se pierde. La sonda s3* al llegar a L2 activa el Paso 2.1 y genera una sonda s'3* hacia L1. De la misma forma s'3* genera s"3* en L1 hacia L4 y la s"3* genera la s"'3* en L4 hacia L3. Finalmente al llegar s"'3* a L3 por el Paso 2.1 se detecta el intebloqueo, siendo elegido el agente a_{23} como víctima. Las sondas emitidas son s1, s'1, s3*, s'3*, s"3*, s"'3*, en total 6 sondas.

La evaluación de la complejidad de un algoritmo de detección distribuido viene determinada por los siguientes criterios [3]: (a) número de mensajes intercambiados para detectar la situación de interbloqueo; (b) longitud de los mensajes; (c) retardo promedio en la detección del interbloqueo.En nuestro algoritmo se pueden distinguir varios casos de formación del circuito. En la formación simultánea de los arcos del circuito, el costo puede variar entre 2n-1, en el mejor caso, y n(n+2)/4, en el peor (supuesto n número par de lugares en interbloqueo).

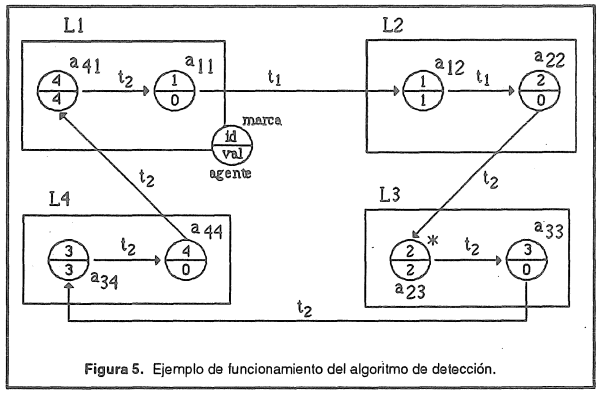


Figura 5. Ejemplo de funcionamiento del algoritmo de detección.

En el caso de la formación secuencial de los arcos del circuito, el costo puede variar entre n, en el mejor caso, y n(n+2)/4, en el peor (supuesto n número par de lugares en el interbloqueo).

El ejemplo corresponde a un caso compuesto (secuencial y simultáneo) de formación del circuito. Son necesarias n= 4 sondas marcadas para detectar el circuito (corresponde al mejor caso secuencial), perdiendose dos sondas no marcadas (corresponde y está acotado por el peor caso simultaneo n(n+2)/4= 4).

Una comparación del algoritmo de detección propuesto con otros algoritmos presentes en la literatura se muestra en la siguiente **tabla 1**.

Algoritmo	Nº de mensajes	Retardo	Tamaño de mens.
Goldman [15]	<m n	t+ nT	variable (medio)
Isloor-Marsland [5]	r (N-1)	0	constante (bajo)
Menasce-Muntz [8]	m (n-1)	nT	variable (bajo)
Obermarck [14]	m (n-1)/2	nT	variable (bajo)
Chandy [12]	<m n	t+ nT	constante (bajo)
Sinha-Natarajan [13]	mejor caso 2 (n-1)	2 (n-1) T	constante (bajo)
	peor caso m (n-1)		
Mittchell-Merritt [9]	m (n-1)/2	(n-1)T	constante (bajo)
Nuestro algoritmo	mejor caso n		
	peor caso n(n+2)/4	nT	constante (bajo)

N: número de lugares; n: número de lugares en el interbloqueo
m: número de procesos en el interbloqueo; T: retardo promedio de comunicación entre lugares; t: retardo en la inicialización del proceso de detección; r: velocidad de actualización del Grafo de estado; d: diámetro del Grafo de estado.

Tabla 1. Comparación de algoritmos de detección distribuidos.

V.- CONCLUSIONES

En este artículo se presenta un algoritmo distribuido de detección del interbloqueo para un Sistema Distribuido de Base de Datos con modelo de ocupación único-recurso. El modelo de la Base de Datos utilizado, basado en un Grafo de Esperas entre Transacciones, es lo suficientemente general como para eludir las descripciones particulares de los mecanismos de control de concurrencia de las transacciones. El algoritmo de detección propuesto sigue la filosofía de los algoritmos arco a cazar y parte de la idea original de Mitchell y Merritt de asociar a cada nodo dos etiquetas una privada y otra pública. El algoritmo presenta las siguientes ventajas: (a) Solo utiliza información local del Grafo de Esperas; (b) todos los

mensajes utilizados en el proceso de detección tienen longitud constante y mínima información; (c) el algoritmo se demuestra formalmente correcto y no genera falsos interbloqueos; (d) por cada interbloqueo sólo un único agente de transacción es responsable de la resolución; (e) el número de mensajes intercambiados para detectar el interbloqueo es menor que el de otros algoritmos de características similares.

REFERENCIAS

[1] E. G. Coffman, M. J. Elphick, A. Shoshani, "Systems Deadlocks", ACM Computing Surveys, vol. 3, no. 2, pp. 67-78, Jun. 1971.

[2] E. Knapp, "Deadlock Detection in Distributed Databases", ACM Computing Surveys, vol. 19, no. 4, pp. 303-328, Dec. 1987.

[3] M. Singhal, "Deadlock Detection in Distributed Systems", IEEE Computer, pp. 37-48, Nov. 1989.

[4] K. P. Eswaran, J. N. Gray, R. A. Loire, I. L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System", Commun. ACM, vol. 19, no. 11, pp. 623-633, Nov. 1976.

[5] S. S. Isloor, T. A. Marsland, "An Effective On-Line Deadlock Detection Technique for Distributed Database Management Systems", Proceedings Computer Science 78, pp. 283-288, Nov. 1978.

[6] P. A. Bernstein, V. Hadzilacos, N. Goodman, "Concurrency Control and Recovery in Database Systems", Adison-Wesley, Reading, Mass. 1987.

[7] V. D. Gligor, S. H. Shattuck, "On Deadlock Detection in Distributed Systems", IEEE Trans. on Softw. Engineering, vol. SE-6, no. 5, pp. 435-440, Sep. 1980.

[8] D.A. Menasce, R.R. Muntz, "Locking and Deadlock Detection in Distributed Databases", IEEE Trans. on Softw. Engineering, vol. SE-5, no. 3, pp. 195-202, May. 1979.

[9] D. P. Mitchell, M. J. Merritt, "A Distributed Algorithm for Deadlock Detection and Resolution", Proc. ACM Conf. Principles of Distributed Computing, pp. 282-284, Aug. 1984.

[10] C. Berge, "Graphs and Hypergraphs", North-Holland Publishing Company, Amsterdam, 1973.

[11] A. Córdoba, "Algoritmos para la Detección del Interbloqueo en Bases de Datos Distribuidas", Tesis Doctoral, Universidad del País Vasco, Leioa 1990.

[12] K. M. Chandy, J. Misra, "A Distributed Algorithm for Detecting Resource Deadlocks in Distributed Systems", Proceedings of the ACM Symposium on Principles of Distributed Computing, pp. 157-164, August 1982.

[13] M. K. Sinha, N. Natarajan , "A Priority-Based Distributed Deadlock Detection Algorithm", IEEE Trans. on Softw. Engineering, pp. 67-80, Jan. 1985.

[14] R. Obermark, "Distributed Deadlock Detection Algorithm", ACM Trans. on Database Systems, vol. 7, no. 2, pp. 187-208, Jun. 1982.

[15] B. Goldman, "Deadlock Detection in Computers Networks", MIT/LCS/TR-185, Laboratory of Computer Science, MIT, Cambridge, Mass., Sep. 1977.

DESIGN OF A PIPELINED MULTIPLIER CHIP AS THE KERNEL OF A VECTOR PROCESSING ARCHITECTURE

DANTE AUGUSTO COUTO BARONE

INSTITUTO DE INFORMÁTICA
UNIV. FEDERAL DO RIO GRANDE DO SUL
CAIXA POSTAL 15064 - CEP 91501-970
PORTO ALEGRE - RS - BRASIL
PHONE: (051) 336-8399 FAX: (051) 336-5576
E-mail: BARONE@INF.UFRGS.BR

ABSTRACT

This paper describes the design of a pipelined multiplier chip, named TREVO. This chip constitutes a major element in a vector processing architecture, which is being researched in the Instituto de Informática of the Federal University of Rio Grande do Sul - Brazil, in the context of the Cp2 project.

KEY WORDS

pipeline - vector architecture - VLSI chip design - multiplier - Cp2

The Cp2 project is funded by CNPq.(Brazilian Council for Research)

I. INTRODUCTION

The huge increase of computer performance nowadays is due to new architectural concepts and to microelectronics technology developments.

In the Instituto de Informática of the Federal University of Rio Grande do Sul (II/UFRGS) is being developed the Cp2 project (Parallel Processing Contributions) which is intented to strongly contribute to the parallel processing area.

One of the major research topics of Cp2 is the development of a vector processing architecture.

The strategy in the design of such computer is to merge advanced research concepts belonging to the Microelectronics and the Computer Architecture fields.

This paper is divided in two main parts:

- the first one describes TREVO: a pipelined 4x4 bits multiplier;
- the second one describes how the TREVO chip is employed in a vector processing architecture:

II. TREVO DESIGN

One of the most important projects under development by the Microelectronics Group (GME) of the II/UFRGS is named CIPREDI (PRE-DIFfUSED INTEGRATED CIRCUITS), which deals with innovative methodologies to the design of gate array circuits, especially sea of gates. CIPREDI's main axis of research concentrate both on VLSI chip design and on CAD tools.

After the design of the complex PEALO circuit (directed memory controller for INTEL 8085 microprocessor), the CIPREDI team judged important to design chips which employ parallel mechanisms.

In this sense, the design of the TREVO chip was launched. This circuit is aimed to perform parallel multiplications, using the pipeline technique. As a first step in this domain and due to silicon area limitations in multiple project-chips runs, a decision was taken to consider only 4x4 bits multiplications: TREVO consists then of a 4x4 bits pipeline combinatorial multiplier chip.

The multiplication algorithm employed by TREVO is showed in figure 1.

				a_3	a_2	a_1	a_0	=A
				b_3	b_2	b_1	b_0	=B
				a_3b_0	a_2b_0	a_1b_0	a_0b_0	=W1
			a_3b_1	a_2b_1	a_1b_1	a_0b_1		=W2
		a_3b_2	a_2b_2	a_1b_2	a_0b_2			=W3
	a_3b_3	a_2b_3	a_1b_3	a_0b_3				=W4
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0	=AxB=P

Fig. 1:TREVO's multiplication algorithm

To produce this multiplication the designers divided the circuit into four stages:

- the first one formed only by and gates, producing intermediate results;
- the second one, formed only by carry save adder (CSA) blocks [1], producing intermediate results;
- the third one, formed only by CSA blocks, producing intermediate results; and

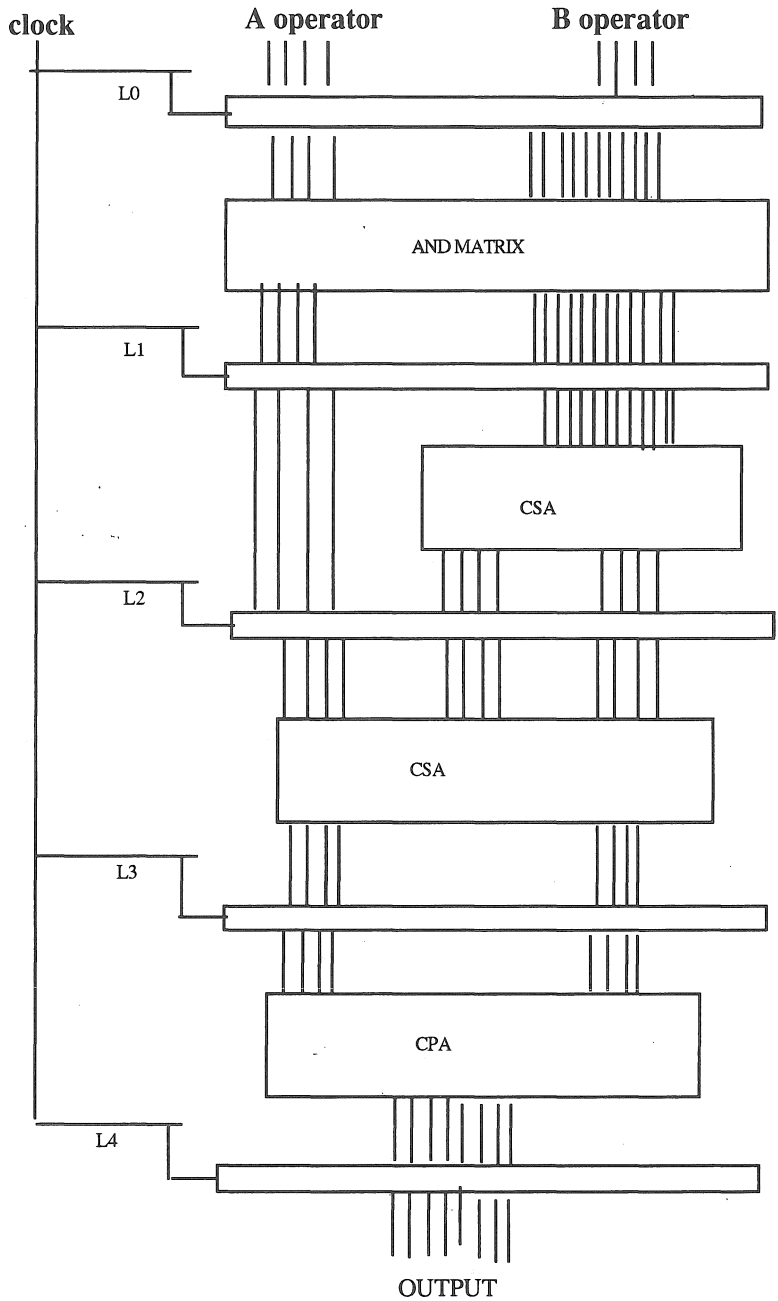


Fig. 2 - TREVO's logic architecture

- the fourth one, formed by carry-propagate adder (CPA) blocks, producing the final results.

The pipeline operation becomes natural, providing parallel execution by the four present stages. The logic diagram of TREVO's architecture is showed in figure 2.

The cells employed by TREVO belong to the CMOS 2,0 mm CIPREDI's library [2]. Basically the chosen cells are: AND, HALF-ADDER, FULL-ADDER and EDGE-SENSITIVE LATCH.

The maximum clock signal frequency imposable to the circuit derives directly from the slowest stage. In this case, as the CPA block (4th stage) presents the highest expected delay (50 ns), this imposes a maximum 20 MHz clock rate.

In each clock pulse each stage's latch capture data from the previous stage. In this sense, the circuit can perform until 20 millions multiplications per second (MOPS),in the case of correct data buffering.

The expected performance of this chip suggests its utilisation in vector operations taking advantage of its high multiplication speed.

III. VECTOR PROCESSING ARCHITECTURE

In the vector processing architecture design the first guideline addressed consistad how to deal withthe 20 MIPS rate, providing to the TREVO chip, 20 MWords/s as input and 20 MWords/s as output, providing then a total 40 MWords/s data rate. The time between words becomes 25 ns, which cannot be supported by a simple main memory, forcing the interleaved acces memory strategy. This leads to buffer the input data or output data allowing the main memory to deal only with read operations or with write operations during fixed slots of time.

Considering that the multiplier pumps 20 MW/s and drives other 20 MW/s, it becomes interesting to design the register system to accomodate vectors.

In this sense the pipeline (the TREVO chip initially) could operate simultaneously with register files: two to receive input data other one to receive the output processed data and another one can be used to receive a new data vector. When the multiplication finishes the pipeline begins to multiply data from an register buffer described above.

This configuration could allow steady maximum speed in the multiplication of large data vectors, even if they are much bugger than the register file length. This implies a minimum of four vector register files. In the computer under development, this number was used.

It must be noted that at each filling or empting of a vector register file the interleaved memory must change from the read to the write condition. This interrupts the data sequence forcing a reinitialization of the memory access. This imposes a speedup diminution equal to the number of memory modules.

For instance if the interleaved memory system disposes of eight modules exchanging data with a N-element register file the final system performance will be of N/N+8, in respect of the ideal performance. Using this relationship different main memory system utilisation efficiences were calculated for different vector regrater leythe (as chawn in table 1).

Vector Register Length (Number of 8 bit registers)	Memory utilization relative efficiency
16	66,6%
32	80%
64	88.8%
128	94.1%
256	96.9%
512	98.4%

Table 1: Memory utilization relative efficiency x Vector Register Length

From this table, it can be visualized that solutions which employ vector registers with lengths higher or equal to 128 elements allow great main memory profit.

In the design it was taken the 256 register lenght and the parameter explaired above of 4 vector register files. This produces a 1K (256 x 4) positions buffer memory.

In this paper the off the shelf circuits employed in the organization of the proposed architecture are not presented, but can be found in [3].

IV. CONCLUSIONS

At a first glance, it can be considered that the designed architecture presents only restricted applicability since it performs only one kind of operation.

Nevertheless, if other operative and logical parts are added to this architecture as pipelined adder/subtractor and a superscalar ULA (towards a VLIW processor), this project could perform many useful vector operations.

ACKNOWLEGMENTS

The author acknowlegdes Mario Magalhães Leboute, Adenilton Ferreira, Leandro Frosi, Gustavo da Paixão and Zhi Wong for their contributions in the developpmnte of this work.

REFERENCES

[1] HWANG, K & BRIGSS ,F .A. Computer Architecture and Parallel Processing. Mcgraw-Hill Bock Company, 1984, U.S.A.

[2] CALAZANS, N.L. et allíi. CMOS 2,0 μm CIPREDI's library. Porto Alegre, Brazil CPGCC/UFRGS. Research Report number 101, 1988.

[3] LEBOUTE, M .M. Vector Processing System Memory Design. Undergraduate Final Work. Departamento de Informática/UFRGS, Porto Alegre, Brazil. 1990.

RESOLUCION DE ECUACIONES EN DERIVADAS PARCIALES MEDIANTE ALGORITMOS PARALELOS

Crespo Fidalgo, J.L. Mora Monte, Eduardo Departamento de Matemática Aplicada y Ciencias de la Computación Universidad de Cantabria Peire Arroba, Juan Departamento de Ingeniería Eléctrica, Electrónica y de Control - UNED.

Resumen

En este artículo se propone una nueva metodología de resolución de ecuaciones en derivadas parciales basada en lo que modernamente se denominan procesos paralelos y distribuidos (PDP), también llamados sistemas complejos y redes neuronales.

Partiendo de una representación en diferencias finitas es posible llegar a un esquema de red en el que las conexiones se programan de antemano. El enfoque es idéntico con cualquier orden que se elija para la discretización por diferencias finitas, lo que da pie a un grado de aproximación variable. La resolución de este esquema en la red resulta equivalente a las técnicas de relajación para sistemas de ecuaciones lineales. Mediante ejemplos simples se ha comprobado la convergencia. Este enfoque permite aprovechar los recursos computacionales típicos de PDP para esta clase de problemas.

Palabras clave: Redes neuronales artificiales, procesamiento paralelo y distribuido, ecuaciones en derivadas parciales, sistemas conexionistas, multiproceso.

1. Introducción

En el Departamento de Matemática Aplicada y Ciencias de la Computación de la Universidad de Cantabria se están estudiando actualmente las posibilidades de los llamados algoritmos de proceso paralelo y distribuido. Concretamente, debido a un mayor conocimiento del área, se ha planteado la posibilidad de aplicarlos a la simulación de sistemas. Aunque este problema puede tratarse en forma completamente estadística (estilo modelos Box-Jenkins) es razonable pensar que un conocimiento añadido pudiera incrementar la bondad de los resultados. Lo que tienen en común los sistemas que son objeto de estudio en la ingeniería es que responden, en su mayoría, a ecuaciones en derivadas parciales, en el caso más general. En este artículo se plantea el primer escalón del estudio de estas aplicaciones, a saber: fijando el tipo de ecuación y el dominio, obtener un sistema que sea capaz de obtener una solución de las ecuaciones de campo en función de las condiciones de contorno.

No tiene sentido en este caso el aprendizaje por ejemplos, tal y como se ha utilizado con frecuencia con estos algoritmos, porque obligaría a resolver muchas veces el problema, con lo cual surge la paradoja de que para resolver el problema, este, se tiene que resolver previamente. En la sección 2 se plantea el origen del método, en la 3 la metodología, en las 4, 5 y 6 se aplica a distintos tipos de ecuaciones y en la 7 se extraen las conclusiones finales así como la línea a seguir en el futuro.

2. Origen del método

Clásicamente los problemas de ecuaciones en derivadas parciales se vienen resolviendo mediante las siguientes técnicas: diferencias finitas, elementos finitos, Montecarlo, Características, etc. Los problemas que se le plantean al ingeniero cuando tiene que resolver un problema son fundamentalmente: precisión requerida, coste computacional y estabilidad. La precisión viene condicionada por el orden del método, con el añadido de que un mayor orden va acompañado de mayor número de operaciones, lo que conduce a un error numérico mayor. Otra opción es mantener el orden pero reducir el tamaño de malla; esto mantiene bajo el error de truncamiento, pero incrementa el coste computacional.

La elección de un método u otro está condicionada por el tipo de ecuación a resolver y la geometría del dominio. Los programas más avanzados suelen emplear el método de elementos finitos, con generación automática de mallado. Esto reduce el esfuerzo del usuario, pero no afecta al coste computacional. La aplicación de las nuevas generaciones de ordenadores a estos problemas se ha realizado planteando métodos eficaces de resolución de sistemas de ecuaciones lineales mediante procesamiento paralelo o vectorial, teniendo en cuenta que casi siempre el problema número final es ese. Obsérvese que, por motivos históricos, se ha partido del método y se ha adaptado a la estructura computacional. Se considera ahora el camino inverso. Se parte de la estructura de resolución en paralelo, que se considera una de las líneas más prometedoras. El sistema más elemental de procesamiento en paralelo es el llamado proceso paralelo y distribuido (también llamados sistemas conectivistas, redes neuronales artificiales, etc.) Consisten en estructuras formadas por elementos interconectados; cada uno tiene varias entradas y una salida. Todo el proceso es local, es decir, un elemento no debe requerir de datos de elementos no conectados a 1. Estos sistemas surgieron tratando de replicar la estructura de las redes de neuronas, si bien de forma muy simplificada. Incluyen una familia muy amplia de algoritmos que se distinguen

unos de otros en la forma de procesar cada elemento y en la forma de conectarse unos con otros. Normalmente los elementos de proceso se constituyen en grupos homogéneos, que pueden distinguirse por uno o ambos aspectos comentados. Cada grupo puede tener el significado de mecanismos de entrada o de salida, estructuras de proceso o de memoria, etc.

En resumen las características deseadas del algoritmo son: - Se ejecuta en un número elevado de procesadores (idealmente). - Los cálculos en cada procesador se realizan sobre variables locales a 1. - La representación final es obra del conjunto de procesadores, por lo que sus "piezas" están distribuidas entre ellos. Con base a los estudios de Hopfield, [5] estos modelos han comenzado a estudiarse desde el punto de vista de la mecánica estadística [2]. Así, se define una función de energía, que depende de la activación o valor de salida de cada elemento (llamado en este contexto el estado), otra de energía libre, entropía. La analogía parte de definir el estado del sistema como el conjunto de los estados de los elementos individuales. La temperatura es en este caso un parámetro que regula el ajuste de cada procesador elemental. Tenemos pues un método que está basado en la idea de procesamiento en paralelo; por tanto, y siempre que se respete su estructura, no tiene que preocuparse por cómo debe adaptarse a este tipo de arquitectura. Por otra parte el hecho de que pueda darse una interpretación mecánica implica que sus valores pueden ser representativos de variables físicas, del tipo de las que intervienen en las ecuaciones que queremos resolver.

3. Metodología

La idea de partida es que cada elemento de la red obtenga el valor de la función desconocida en un punto, y que la estructura de la función de activación y las conexiones sea tal que se verifique la ecuación diferencial que intentamos resolver. A continuación se desarrolla esta idea.

Se plantea una ecuación:

$Df=0$

siendo D un operador diferencial genérico.

Si se toma una aproximación del operador diferencial, se tendrá una ecuación en diferencias que relaciona el valor de f en distintos valores de sus variables:

$L(f(x_1),f(x_2),...)=0$

donde L es un operador que aproxima D y x_1, x_2, ... son vectores de las variables en distintos puntos.

Si en este caso se supone que f es calculada por nuestros procesadores, se puede asignar a cada uno el valor de f en cada uno de los puntos ($f(x_i)$) (ver fig. 1) y obligar a que la estructura de la red, es decir las conexiones, sean tales que se verifique la ecuación anterior.

Concretamente, si L es una aproximación local y si f es despejable, la ecuación se podrá poner como:

f(centro)=G(f(x(1)),f(x(2)),...)

y esta misma ecuación ser válida en cada punto, tomando cada vez la aproximación local correspondiente.

Este planteamiento es totalmente análogo al desarrollado por el método de diferencias finitas, donde se plantean aproximaciones lineales de los operadores diferenciales.

Si el sistema de ecuaciones lineales resultante se resuelve siguiendo una vía iteractiva (Gauss-Seidel, por ejemplo) se puede llegar a la misma expresión anterior.

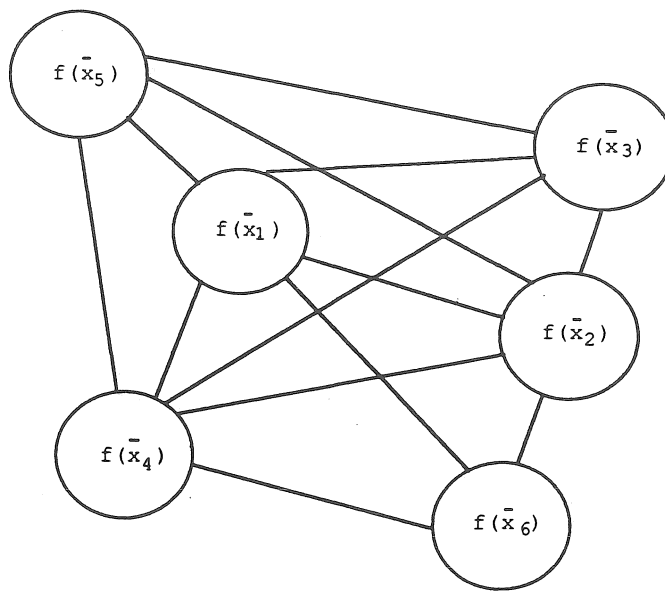


Fig. 1. Representación de un campo mediante una red de procesadores. Cada elemento calcula el valor de la función en un punto.

No obstante en la presentación que se ha hecho no se ha requerido que la aproximación fuera lineal, lo que le da un carácter más general. En concreto los requisitos para llegar a la última expresión son:

- Un operador en diferencias que aproxime el operador diferencial localmente.
- El valor de la función en el centro de este entorno local debe poderse despejar explícitamente.
- El sistema formado por todas estas ecuaciones ha de ser estable.

El último requisito es meramente de resolución, para garantizar que a partir de una configuración cualquiera del sistema se puede llegar a la solución.

4. Aplicación a ecuaciones elípticas

Sea la ecuación $\Delta f = 0$ en un dominio 1-D.

Tomemos una aproximación por diferencias finitas de primer orden

$$f(x-\Delta)+f(x+\Delta)-2f(x)=0$$

Tomando una discretización con intervalo constante, se tiene para un nodo f_i

$$f_i = \frac{f_{i-1}+f_{i+1}}{2} = 0'5f_{i-1} + 0'5f_{i+1}$$

Sumando f_i a ambos lados y dividiendo por dos, para conseguir estabilidad se tiene:

$$f_i = 0'25f_{i-1} + 0'25f_{i+1} + 0'5f_i$$

donde el término f_i de la derecha corresponde al valor previo de la interacción.

Analíticamente se tiene

$$\frac{d^2f}{dx^2}=0 \Rightarrow \frac{df}{dx}=K_1 \Rightarrow f=K_1x+K_2$$

Luego la solución de un recta que une las condiciones de contorno, por ejemplo el caso presentado en la fig. 2:

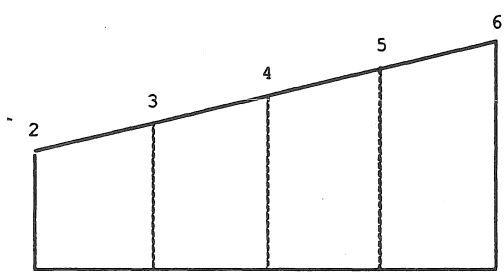


Fig. 2 Ejemplo elemental de ecuación diferencial de 2º orden resuelto analíticamente.

Los resultados de la red, programada según la expresión deducida anteriormente se presetan en la fig. 3:

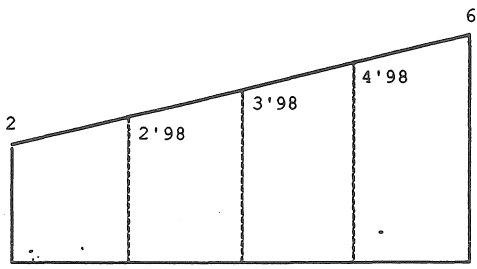


Fig. 3. Solución obtenida por la red para el mismo problema de la fig. 2.

Por lo que se deduce que la precisión está limitada sólo por el número de iteraciones.

Se tiene ahora un caso más complicado como el que se presenta con la ecuación

$$\frac{d}{dx}\left(K\frac{df}{dx}\right)+\frac{d}{dy}\left(K\frac{df}{dy}\right)=0 \text{ con } K=K(f)$$

aproximándola como:

$$\frac{\frac{1}{2}\left\{K\left[T(x_1,y_0)\right]+K\left[T(x_0,y_0)\right]\right\}\frac{T(x_1,y_0)-T(x_0,y_0)}{x_1,x_0}-\frac{1}{2}\left\{K\left[T(x_{-1},y_0)\right]+K\left[T(x_0,y_0)\right]\right\}}{\frac{1}{2}(x_0+x_1)-\frac{1}{2}(x_1+x_0)}$$

$$\frac{T(x_0,y_0)-T(x_{-1},y_0)}{x_0-x_{-1}}+\frac{\frac{1}{2}\left\{K\left[T(x_0,y_1)\right]+K\left[T(x_0,y_0)\right]\right\}\frac{T(x_0,y_1)-T(x_0,y_0)}{y_1-y_0}-}{}$$

$$\frac{-\frac{1}{2}\left\{K\left[T(x_0,y_{-1})\right]+K\left[T(x_0,y_0)\right]\right\}\frac{T(x_0,y_0)-T(x_0,y_{-1})}{y_0,y_{-1}}}{\frac{1}{2}(y_0+y_1)-\frac{1}{2}(y_{-1}+y_0)}=0$$

Se puede despejar, haciendo $\Delta x=\Delta y=cte$

$$T(x_0,y_0)=\frac{\left\{K\left[T(x_1,y_0)\right]+K\left[T(x_0,y_0)\right]\right\}T(x_1,y_0)+\left\{K\left[T(x_{-1},y_0)\right]+K\left[T(x_0,y_0)\right]\right\}T(x_1,y_0)+}{K\left[T(x_1,y_0)\right]+K\left[T(x_1,y_0)\right]+K\left[T(x_0,y_1)\right]+K\left[T(x_0,y_{-1})\right]+4K\left[T(x_0,y_0)\right]}$$

$$\underline{\left\{K\left[T(x_0,y_1)\right]+K\left[T(x_0,y_0)\right]\right\}T(x_0,y_1)+\left\{K\left[T(x_0,y_{-1})\right]+K\left[T(x_0,y_0)\right]\right\}T(x_1,y_{-1})}$$

A esta ecuación se le aplicaría el método iterativo del caos anterior.
En la fig. 4 pueden verse los resultados para un dominio y condiciones de contorno concretas

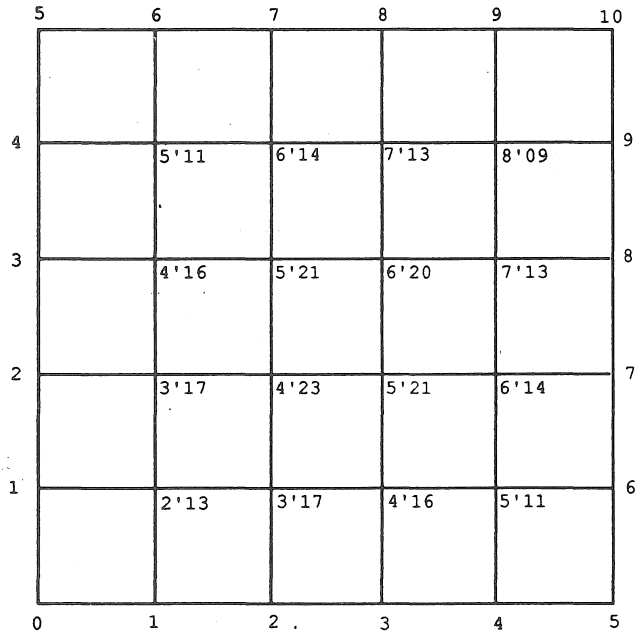


Fig. 4 Resolución de ecuación elíptica no lineal.

5. Aplicación a ecuación parabólica

Sea $\frac{d^2f}{dx^2} = K\frac{df}{dt}$ con $K = \frac{K_0}{f}$

Utilizando la aproximación de Crank-Nicolson con proyección hacia adelante se llega a una expresión de la forma

$$f(x_0,t_1)^j = \beta\frac{\alpha\,\varepsilon\,\omega_1\,f(x_1,t_j)+Kf(x_0,t_0)}{K+\omega_0\,\alpha}+(1-\beta)f(x_0,t_1)^{j-1}$$

donde el superíndice j se refiere a la interación, β es el parámetro a elegir para las iteraciones, $\alpha = \frac{\Delta t}{\Delta x}$, ω_j son múltiplos enteros de $\frac{1}{\Delta x}$

Esto se puede solucionar para unas condiciones iniciales y en los extremos, por ejemplo, para:

$$K=\frac{10}{f},\quad \alpha=0'01,\ \Delta x=0'02,\ \beta=0'5$$

Se obtendrían los resultados incicados en la fig. 5.

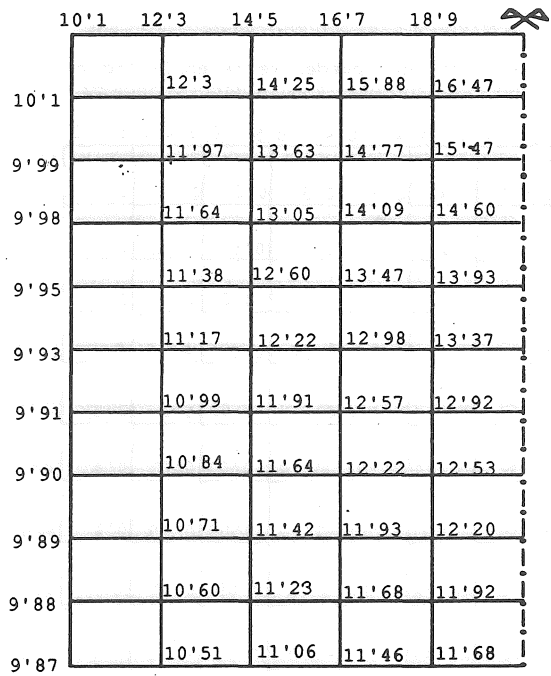


Fig. 5. Resolución de una ecuación parabólica no lineal con condiciones simétricas.

En el sistema hay en cada momento una capa para el instante anterior y otra para el actual

6. Aplicación a ecuación hipérbole

Sea $\dfrac{d^2f}{dx^2}=\dfrac{d^2f}{dt^2}$

Suponiendo $\Delta x=\Delta t=$ cte se puede aproximar como

$$f(x_1,t_0)-f(x_0,t_0)-\left[f(x_0,t_0)-f(x_1,t_0)\right]=f(x_0,t_1)-f(x_0,t_0)-\left[f(x_0,t_0)-f(x_1,t_{-1})\right]$$

de donde se puede despejar $f(x_0, t_1)$ y resolver por relajación como en los casos anteriores.

La red tendrá 3 capas: una para t_{-1}, otra para t_0 y otra para t_1.
Tal y como se ha planteado, la aproximación está en el límite de estabilidad.

Los resultados para un caso concreto pueden verse en la figura 6:

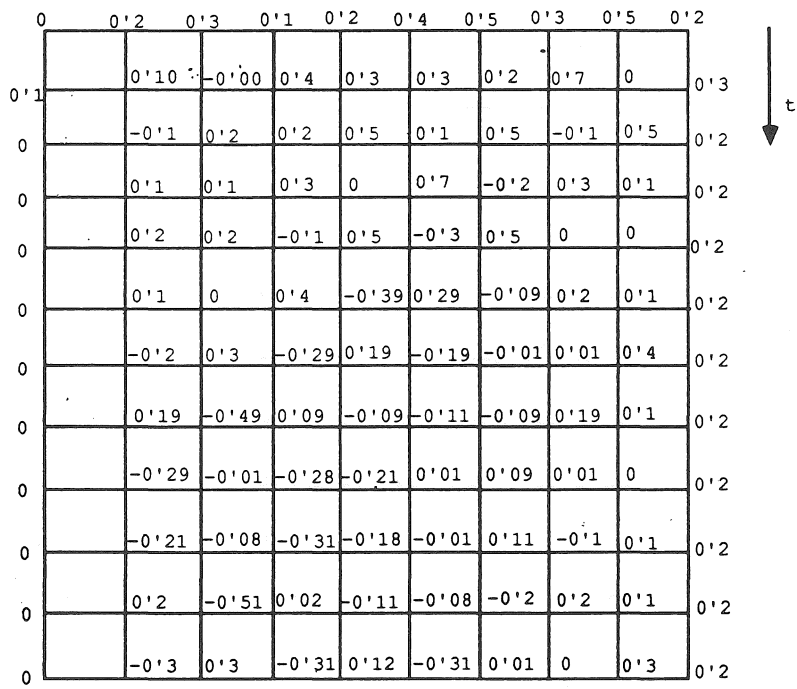


Fig. 6 Resolución de ecuación hiperbólica sencilla.
El paso de tiempo utilizado es el máximo estable.

7. Conclusiones

Se ha presentado un enfoque que permite compaginar los algoritmos de proceso paralelo y distribuido con la resolución de ecuaciones en derivadas parciales en paralelo. Esta idea, inspirada en la analogía entre elementos de proceso y partículas, tal y como se estudian en Mecánica Estadística, puede presentar un amplio campo de aplicación. Los casos prácticos aquí planteados tienen como caso particular la resolución iteractiva de aproximaciones en diferencias finitas, lo que se ha demostrado con varios ejemplos. Ha quedado patente la

forma de plantear la analogía mencionada, y se ha comprobado, a nivel de los ejemplos, la utilidad del método que permite sustituir la programación específica de ordenadores multiproceso por la de un sistema PPD. Con el desarrollo de los nuevos ordenadores basados en técnicas del PDP, se abrirían muevos campos de aplicación y en especial en los problemas más complejos dentro del ámbito de la ingeniería.

Referencias

[1] J.L. McClelland y D.E. Rumelhart, Explorations in Parallel Distributed Processing, MIT Press, 1989.

[2] J. Hertz, A. Krogh y R. G. Palmer, Introduction to the theory of neural computation, Addison-Wesley, 1991.

[3] M. Quinn. Designing Efficient Algorithms for Parallel Computers. McGraw-Hill, 1987.

[4] J. M. Ortega, R.G. Voigt, Solution of Partial Differential Equations on Vector and Parallel Computers, SIAM, 1985.

[5] Hopfield, J.J. (1982). Neural Networks and Physical Systems with Emergent Colective Computational Abilities. Proceedings of the National Academy of Sciences, USA 79, 2554-2558. Reprinted in Anderson and Rosenberg [1988].

CSP Probabilístico (PCSP): Un modelo operacional

F. Cuartero Gómez
Dpto.Informática
Esc. Politécnica
Univ. Castilla-La Mancha
Albacete, SPAIN 02071
Fax: 34 67 506650
utfcuartero@02ccv1.ucma.es

D. de Frutos Escrig
Dpto.Informática y Automática
Fac. Matemáticas
Univ. Complutense
Madrid, SPAIN 28040
Fax: 34 1 5439489
W575 @ EMDUCM11.bitnet

V. Valero Ruiz
Dpto.Informática
Esc. Politécnica
Univ. Castilla-La Mancha
Albacete, SPAIN 02071
Fax: 34 67 506650
utvalero@02ccv1.ucma.es

Resumen

Presentamos un algebra de procesos para la modelización de procesos comunicantes probabilísticos. El tipo de álgebra es asíncrono, al igual que los modelos clásicos, y contrariamente a la mayoría de álgebras probabilísticas estudiadas hasta el momento (PCCS), que son síncronas. La principal dificultad en nuestro modelo proviene de disponer en el mismo de dos tipos de elección (interna y externa), al contrario que en los anteriores, donde sólo existe uno. Otra dificultad, que nuestro modelo no alcanza a resolver es el funcionamiento del operador de ocultamiento. Si queremos mantener su significado clásico, necesitaríamos añadir más potencia a nuestra semántica. Definiremos una semántica operacional del lenguaje, y la usaremos para razonar sobre su potencia mediante un ejemplo no trivial, en el que mostraremos la posibilidad de modelizar propiedades como la seguridad (*reliability*), y el cálculo de tiempos medios (*performance*).

1 Introducción

Los algoritmos aleatorizados están siendo utilizados cada vez con mayor asiduidad. En particular ya han comenzado a ser utilizados en la ya de por sí compleja programación de sistemas distribuidos (veáse [11]). El uso de estos algoritmos redunda en una mayor velocidad y simplificación de las soluciones. Incluso, en ocasiones no existe alternativa a la aleatorización de las soluciones. Por tanto, los formalismos matemáticos utilizados en la especificación de sistemas en los que aparecen comportamientos aleatorios debe incluir componentes probabilísticas. Los mismos permitirán en particular estudiar la corrección de dichos sistemas, que a diferencia de lo que sucede en el caso determinista, no vendrá determinada por alguna propiedad que sólo pueda ser cierta o falsa, sino que puede venir determinada por la probabilidad de que una cierta propiedad se cumpla.

El lenguaje *Communicating Sequential Processes (CSP)* [5] ofrece un formalismo adecuado para la especificación de sistemas distribuidos no deterministas. Sus principales ventajas radican en servir como soporte de un razonamiento algebraico efectivo en el tratamiento de procesos concurrentes. Nuestra intención en el presente trabajo es construir una versión probabilística lo más sencilla posible de CSP que combine toda su potencia (al menos en lo que se refiere al tratamiento de la elección) con las nociones

probabilísticas de corrección (que también pueden extenderse a la terminación). Este lenguaje lo hemos denominado CSP probabilístico (PCSP).

2 Sintaxis de PCSP

La sintaxis de PCSP es una sencilla variante de la de CSP, con la única diferencia de añadir una distribución de probabilidades a todos aquellos operadores que conllevan una elección, interna o externa. La resumimos mediante la expresión BNF

$$P ::= STOP \mid X \mid a \rightarrow P \mid [p_1]P \sqcap [p_2]P \mid [p_1]P \,\Box\, [p_2]P$$

$$\mid [p_1]P \parallel_A [p_2]P \mid P\backslash(a,p) \mid \mu X.P$$

donde $p \in [0,1]$, p_1, $p_2 \in [0,1]$, con $p_1 + p_2 = 1$ y $X \in Idf$.

El significado intuitivo de cada uno de los operadores es el siguiente:

STOP y PREFIJO: STOP es el proceso que simboliza una máquina rota, incapaz de realizar ninguna acción. Dados un proceso P y una acción a, $a \rightarrow P$ es un proceso que ejecuta en primer lugar la acción a, y después pasa a comportarse como el proceso P.

ELECCIÓN INTERNA: Dados dos procesos P y Q, y $p_1, p_2 \in [0,1]$ tales que $p_1 + p_2 = 1$, entonces $[p_1]P \sqcap [p_2]Q$ es un proceso que se comporta como P con probabilidad p_1 o como Q con probabilidad p_2.

ELECCIÓN EXTERNA: Dados dos procesos P y Q, y $p_1, p_2 \in [0,1]$ tales que $p_1 + p_2 = 1$, entonces $[p_1]P \,\Box\, [p_2]Q$ es un proceso cuyo comportamiento es principalmente determinista. Si el entorno del proceso decide ejecutar una acción que puede ser ejecutada por P o Q, pero no por ambos, será ejecutada por el proceso correspondiente con la probabilidad con la que este proceso puede ejecutar la acción, continuando después en consecuencia.

Sólo en el caso de que ambos puedan realizar dicha acción la elección será aleatoria, de acuerdo a las probabilidades p_1 y p_2. Sin embargo, aún en este caso, la elección externa no sería equivalente a la interna, ya que además de las dos probabilidades citadas hay que tener en cuenta las probabilidades que cada proceso tiene de aceptar la ejecución de dicha acción. Así, la probabilidad de ejecutar una acción es mayor en el proceso resultante de elegir entre dos, que en cualquiera de ellos.

COMPOSICIÓN PARALELA: Dados dos procesos P y Q, $p_1, p_2 \in [0,1]$ tales que $p_1 + p_2 = 1$ y un conjunto de acciones A, entonces $[p_1]P \parallel_A [p_2]Q$ es un proceso que corresponde a la ejecución concurrente de P y Q, sincronizando sobre las acciones de A. Dada una acción $a \in A$, para que ésta pueda realizarse, ambos procesos deben estar en condiciones de ejecutarla, obteniéndose como resultado de esa ejecución una única acción a. Para las acciones no contenidas en el conjunto A de sincronización, los procesos evolucionan independientemente, salvo que ambos puedan realizar la citada acción, en cuyo caso se decide a quién corresponde la ejecución atendiendo a las probabilidades p_1 y p_2, de la misma forma que en la elección externa.

OCULTACIÓN: Dado un proceso P y dado un par (a,q); con $a \in \Sigma$ y $q \in [0,1]$, entonces $P\backslash(a,q)$ es un proceso en el que las ejecuciones de la acción oculta a no pueden ser observadas. La probabilidad q tiene por objeto decidir sobre las elecciones externas en el caso de que aparezca la acción a como una de las posibles opciones. En este caso, el entorno no siempre tiene la posibilidad de elegir, porque puede ocurrir que el sistema

decida autónomamente ejecutar la acción oculta. Esta elección entre la acción oculta y las observables será entonces aleatoria, y la probabilidad con la que el sistema decidirá elegir la acción oculta será q.

RECURSIÓN: La recursión nos permite expresar patrones de comportamiento para procesos infinitos. Podemos ver la semántica de este operador como la solución de un sistema de ecuaciones mutuamente recursivas, donde los identificadores representan los procesos incógnitas.

3 Semántica operacional de PCSP

La semántica que presentaremos, y que posteriormente utilizaremos para razonar sobre los comportamientos de los procesos, será una semántica operacional. Así, vamos a definir la evolución o ejecución de un proceso mediante un sistema probabilístico de transiciones (veáse [7]), donde las transiciones serán aquellas generadas por un conjunto de reglas, cada una de las cuales contribuye a definir la semántica de alguno de los operadores del lenguaje. Seguiremos el estilo estructurado de Plotkin [10] y Milner [9].

Definición 1 Una *transición probabilística observable* es una tupla $\langle P, Q, a, p\rangle$ donde P y Q son procesos, $a \in \Sigma$, y $p \in (0,1]$. Usualmente representaremos una transición de la forma siguiente:

$$P \xrightarrow{a}_p Q$$

□

Seguidamente, presentamos y comentamos las reglas de transición correspondientes al conjunto de operadores del lenguaje.

1) $\dfrac{}{a \to P \xrightarrow{a}_1 P}$

2) $\dfrac{P \xrightarrow{a}_p Q}{[q]P \sqcap [1-q]R \xrightarrow{a}_{pq} Q}$

3) $\dfrac{P \xrightarrow{a}_p Q}{[q]R \sqcap [1-q]P \xrightarrow{a}_{p(1-q)} Q}$

4) $\dfrac{P \xrightarrow{a}_p P',\ Q \xrightarrow{a}_q}{[r]P \,\square\, [1-r]Q \xrightarrow{a}_s P'}$
donde $s = pqr + p(1-q)$

5) $\dfrac{P \xrightarrow{a}_p ,\ Q \xrightarrow{a}_q Q'}{[r]P \,\square\, [1-r]Q \xrightarrow{a}_s Q'}$
donde $s = pq(1-r) + (1-p)q$

6) $\dfrac{P \xrightarrow{a}_p P',\ Q \xrightarrow{a}_{p'},\ a \notin A}{[q]P \parallel_A [1-q]Q \xrightarrow{a}_s [q]P' \parallel_A [1-q]Q}$
donde $s = pp'q + p(1-p')$

7) $\dfrac{P \xrightarrow{a}_{p'},\ Q \xrightarrow{a}_p Q',\ a \notin A}{[1-q]P \parallel_A [q]Q \xrightarrow{a}_s [1-q]P \parallel_A [q]Q'}$
donde $s = pp'q + p(1-p')$

8) $\dfrac{P \xrightarrow{a}_p P',\ Q \xrightarrow{a}_{p'} Q',\ a \in A}{[q]P \parallel_A [1-q]Q \xrightarrow{a}_{pp'} [q]P' \parallel_A [1-q]Q'}$

9) $\dfrac{P[P/X] \xrightarrow{a}_p P'}{\mu X.P \xrightarrow{a}_p P'}$

donde $P[P/X]$ representa la sustitución sintáctica en P del identificador X por el proceso P.

Observese que en algunas de estas reglas aparece la noción derivada $P \xrightarrow{a}_p$ cuya definición se incluye a continuación.

Definición 2 i) Llamamos semántica operacional del lenguaje PCSP (sin operador de ocultamiento) al multiconjunto de transiciones probabilísticas que pueden ser inferidas con el sistema de reglas anterior, y donde cada transición aparece tantas veces como formas diferentes haya de derivarla.

ii) Sea a una acción y P un proceso. Diremos que p es la probabilidad con la que P ejecuta la acción a, lo que representaremos de la forma

$$P \xrightarrow{a}_{p}$$

si $p = \sum_{i \in I} p_i$ donde I es el multiconjunto de transiciones

$$P \xrightarrow{a}_{p_i} P_i$$

que ejecutan la acción a, que pueden ser derivadas usando las reglas. □

Las tres primeras reglas son suficientemente claras y no merecen más comentario. En las reglas cuarta y quinta, la probabilidad de que el proceso $[r]P \,\square\, [1-r]Q$ ejecute la acción a y pase a comportarse como P', o Q', según cada transición generada, se calcula considerando la posibilidad de que o bien P o bien Q ejecuten dicha acción. Sin embargo, si ambos procesos pueden ejecutar dicha acción, sólo uno de ellos podrá ser elegido para hacerlo, y esta elección se hará teniendo en cuenta la probabilidad r indicada en la elección externa, que define la probabilidad con la que seleccionaremos el primer proceso cuando esta situación ocurre.

Las reglas sexta y séptima cubren el caso de *interleaving*. Con ellas se describe que si la acción a ejecutar no pertenece al conjunto de sincronización, será realizada por el proceso componente que esté capacitado para ello. Si ambos procesos lo están, se elegirá aleatoriamente, de manera análoga a como se ha descrito para el operador de elección externa, atendiendo a la probabilidad asociada al operador composición paralela, que también indica la probabilidad con la que será seleccionada en tales situaciones la primera componente.

La octava regla describe el caso en que la acción pertenece al conjunto de sincronización. Deseamos entonces que la acción sea ejecutada por ambos procesos simultáneamente, y ésto, obviamente ocurrirá con una probabilidad igual al producto de las correspondientes a las transiciones combinadas de los procesos componentes.

Finalmente, la última regla trata el operador de la recursión en la forma tradicional, no necesitando por ello de ningún comentario adicional.

Veamos ahora porqué es necesario trabajar con multiconjuntos al definir la semántica operacional del lenguaje. Consideremos al respecto el proceso $P = [\frac{1}{2}]a \to STOP \sqcap [\frac{1}{2}]a \to STOP$. Tenemos que

$\mathcal{T}_1 = \; P \xrightarrow{a}_{\frac{1}{2}} STOP$ es derivable aplicando las reglas 1 y 2, mientras
$\mathcal{T}_2 = \; P \xrightarrow{a}_{\frac{1}{2}} STOP$ lo es aplicando las reglas 1 y 3.

Con ambas derivaciones se infiere que P puede ejecutar la acción a con probabilidad $\frac{1}{2}$. Sin embargo, parece necesario que quede constancia de que se trata de dos computaciones diferentes, a fin de que pueda inferirse a la postre que efectivamente la acción a será ejecutada con probabilidad 1.

Sin embargo, hemos preferido evitar una formalización explícita del multiconjunto de transiciones derivadas como el utilizado en [3], donde las transiciones están indexadas con índices que indican la vía en que han sido generadas, para no perder al lector con un tecnicismo adicional, que no añade ninguna idea nueva.

Comentamos por último el tema del operador de ocultación. Para el mismo no hemos presentado regla alguna, lo que naturalmente implica que no ha quedado definida su semántica. Ello se debe a que lamentablemente no es posible capturar su comportamiento deseado, en el marco de una semántica que razone en base al estudio de una acción por vez.

El problema radica en el hecho de que al aplicar el operador de ocultamiento sobre una elección externa, ésta se convierte en parcialmente interna. En cambio el significado de las elecciones internas no varía al aplicarles un ocultamiento. Consideremos al respecto el siguiente

Ejemplo 1 Sean P y Q los siguientes procesos:

$$\begin{aligned} P = \ & [\tfrac{1}{2}]([\tfrac{1}{2}]a \to STOP \Box [\tfrac{1}{2}]b \to STOP) \sqcap [\tfrac{1}{2}]STOP \\ Q = \ & [\tfrac{1}{2}]a \to STOP \sqcap [\tfrac{1}{2}]b \to STOP \end{aligned}$$

Ambos procesos son equivalentes respecto de cualquier noción basada en la presente semántica operacional, pues la misma es idéntica para ambos. Sin embargo, si ocultamos la acción a el comportamiento deseado pasa a ser diferente, pues en $P\backslash(a,q)$ la probabilidad de ejecutar la acción b disminuye, mientras que en $Q\backslash(a,q)$ sigue siendo $\frac{1}{2}$, lo que es imposible de lograr si efectivamente las semánticas de P y Q son idénticas. □

4 Computaciones de procesos probabilísticos

En la sección anterior, hemos introducido el conjunto de reglas que definen las transiciones de los procesos probabilísticos. No obstante, las transiciones sólo definen el primer paso de la evolución de un proceso. Por ello, necesitamos un razonamiento más potente, encadenando transiciones para adentrarnos en la ejecución de los procesos.

Definición 3 Llamamos computaciones de un proceso probabilístico P a las secuencias de transiciones:

$$P = P_0 \xrightarrow{a_1}_{p_1} P_1, \ \ldots\ , P_{n-1} \xrightarrow{a_n}_{p_n} P_n$$

incluídas en la semántica operacional de PCSP. Usualmente las representaremos en la forma

$$\mathcal{C} = P \xrightarrow{a_1}_{p_1} P_1 \xrightarrow{a_2}_{p_2} \ldots \xrightarrow{a_n}_{p_n} P_n$$

Llamaremos una *derivada* de P generada por $\mathcal{C}$ al triple $\langle s, P_n, p\rangle$, donde $s = a_1 \ldots a_n$ y $p = p_1 \cdots p_n$. Usualmente representaremos estas derivadas con la notación:

$$P \overset{s}{\Longrightarrow}_p P_n$$

De nuevo cada proceso tiene un multiconjunto de computaciones, en el que cada computación aparece tantas veces como indica el producto de las veces que aparecen las componentes en la semántica del lenguaje. □

La noción de derivada puede ser fácilmente formalizada en la forma siguiente:

Definición 4 Sea s una secuencia (posiblemente vacía) de acciones, $p \in (0,1]$ y P, Q dos procesos. Entonces, $P \overset{s}{\Longrightarrow}_p Q$ es una computación si y sólo si es posible inferirla por medio del uso de las reglas:

1. $P \overset{\langle\rangle}{\Longrightarrow}_1 P$
2. Si $P \xrightarrow{a}_p P'$ y $P' \overset{s}{\Longrightarrow}_q Q$, entonces $P \overset{as}{\Longrightarrow}_{pq} Q$

□

5 Ejemplo. Protocolo AUY

Presentaremos un ejemplo en el cual modelamos el protocolo AUY (ver [1]) utilizado en la transmisión de datos, que asegura una transmisión segura sobre un sistema en el que los canales de comunicación pueden fallar. Este ejemplo también se ha estudiado en [3], sobre el marco del álgebra de procesos (PCCS) allí presentado. Sin embargo, en nuestra opinión, este tipo de álgebra síncrona plantea algunos problemas que dificultan el modelado de sistemas. Unos derivados de la sincronía, pues es necesario sincronizar *a mano* todas las componentes del sistema que no están sincronizadas de una manera natural, y otros, derivados de la inexistencia de un operador explícito de elección externa, que sólo puede ser parcialmente simulado por medio de una adecuada combinación de las probabilidades en el único operador de elección que incluye, junto con la restricción de los procesos a un adecuado conjunto de acciones observables. Este mecanismo no es muy intuitivo, puesto que un cambio en las probabilidades, o una restricción a un conjunto inadecuado de acciones, implicaría la desaparición de la posibilidad de esta interpretación de la elección como externa. Por otra parte, no resulta en absoluto evidente generalizar esta simulación a una ocurrencia cualquiera de la elección externa.

En nuestro lenguaje estos problemas desaparecen, puesto que disponemos de un operador paralelo asíncrono, y un operador explícito de elección externa, ambos con una semántica adecuada.

Por simplicidad, omitiremos a lo largo del ejemplo las probabilidades en las apariciones de los operadores de elección externa y composición paralela, puesto que de hecho no son necesarias, ya que nunca aparecerán elecciones entre procesos que puedan ejecutar como primera acción una misma.

El sistema consiste de dos agentes, un transmisor y un receptor, que se comunican mediante mensajes. El transmisor envía un mensaje al receptor usando un canal no seguro, que puede perderlo con una probabilidad conocida p. Suponemos que si el mensaje original se pierde durante la transmisión, el canal lo reemplazará con un mensaje nulo λ, que enviará al receptor. Este tomará el mensaje de su canal de entrada, y si no es λ, lo enviará al exterior, devolviendo un mensaje de reconocimiento al transmisor que enviará por un segundo canal. De nuevo, el canal podrá perder el mensaje (suponemos que con la misma probabilidad p, aunque esta suposición no es necesaria), en cuyo caso el transmisor recibirá un mensaje nulo λ'. Si el transmisor recibe esta señal λ', repetirá la emisión, enviando de nuevo el mensaje original. Finalmente, cuando el transmisor reciba la señal de reconocimiento podrá comenzar un nuevo ciclo.

El proceso PCSP que modela este protocolo es el proceso P definido como sigue:

Transmisor:

$$T = in \rightarrow T_1$$
$$T_1 = msg \rightarrow (ack \rightarrow T \,\Box\, \lambda' \rightarrow T_1)$$

Canal Primero:

$$C_1 = msg \rightarrow ([p]\lambda \rightarrow C_1 \sqcap [1-p]msg' \rightarrow C_1)$$

Canal Segundo:

$$C_2 = (ack' \rightarrow ([p]\lambda' \rightarrow C_2 \sqcap [1-p]ack \rightarrow C_2)) \Box (\lambda'' \rightarrow \lambda' \rightarrow C_2)$$

Receptor:

$$R = (msg' \rightarrow out \rightarrow ack' \rightarrow R) \Box (\lambda \rightarrow \lambda'' \rightarrow R)$$

Protocolo AUY:

$$P = T \|_{\{msg,ack,\lambda'\}} ((C_1 \|_{\emptyset} C_2) \|_{\{msg',ack',\lambda,\lambda''\}} R)$$

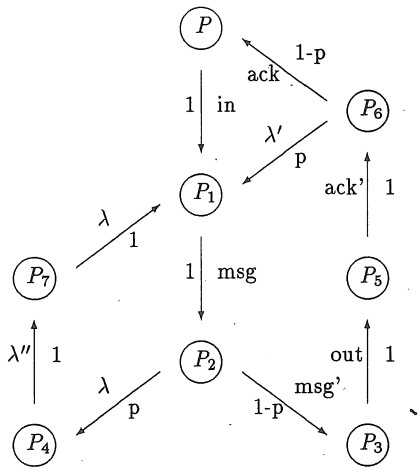


Figura 1: Sistema Probabilístico de Transiciones para el Protocolo AUY

El comportamiento de este proceso se muestra en el sistema probabilístico de transiciones representado en la figura 1. En donde:

$$
\begin{aligned}
P_1 &= T_1 \|_{\{msg,ack,\lambda'\}}((C_1 \|_{\emptyset} C_2) \|_{\{msg',ack',\lambda,\lambda''\}} R) \\
P_2 &= (ack \rightarrow T \Box \lambda' \rightarrow T_1) \|_{\{msg,ack,\lambda'\}}((([p]\lambda \rightarrow C_1 \sqcap [1-p]msg' \rightarrow C_1) \|_{\emptyset} C_2) \\
&\quad \|_{\{msg',ack',\lambda,\lambda''\}} R) \\
P_3 &= (ack \rightarrow T \Box \lambda' \rightarrow T_1) \|_{\{msg,ack,\lambda'\}}((C_1 \|_{\emptyset} C_2) \|_{\{msg',ack',\lambda,\lambda''\}} (out \rightarrow ack' \rightarrow R)) \\
P_4 &= (ack \rightarrow T \Box \lambda' \rightarrow T_1) \|_{\{msg,ack,\lambda'\}}((C_1 \|_{\emptyset} C_2) \|_{\{msg',ack',\lambda,\lambda'\}} (\lambda'' \rightarrow R)) \\
P_5 &= (ack \rightarrow T \Box \lambda' \rightarrow T_1) \|_{\{msg,ack,\lambda'\}}((C_1 \|_{\emptyset} C_2) \|_{\{msg',ack',\lambda,\lambda''\}} (ack' \rightarrow R)) \\
P_6 &= (ack \rightarrow T \Box \lambda' \rightarrow T_1) \|_{\{msg,ack,\lambda'\}}((C_1 \|_{\emptyset} ([p]\lambda' \rightarrow C_2 \sqcap [1-p]ack \rightarrow C_2)) \\
&\quad \|_{\{msg',ack',\lambda,\lambda''\}} R) \\
P_7 &= (ack \rightarrow T \Box \lambda' \rightarrow T_1) \|_{\{msg,ack,\lambda'\}}((C_1 \|_{\emptyset} (\lambda'' \rightarrow C_2)) \|_{\{msg',ack',\lambda,\lambda''\}} R)
\end{aligned}
$$

Nuestro lenguaje nos permite razonar sobre diferentes propiedades del protocolo, tales como la fiabilidad (*reliability*) y el tiempo esperado de transmisión. Por ejemplo, es sencillo comprobar que la probabilidad con la que un ciclo termina tras un bucle simple es $(1-p)^2$. Sin embargo, el hecho más importante es que todos los elementos de la modelización del sistema son muy naturales, y no hemos necesitado ningún truco como el que aparece (y necesariamente!) en el modelo presentado en [3].

Para finalizar, formalizaremos seguidamente la prueba de la seguridad del protocolo (cualquier mensaje será transmitido, y el reconocimiento se recibirá con probabilidad 1), y calcularemos la longitud media de un ciclo de transmisiones.

Cualquier computación de nuestro sistema comenzará con la transición

$$P \xrightarrow{in}_1 P_1$$

y un ciclo terminará cuando el estado P se recupere.

Desde P_1 tenemos las siguientes computaciones correspondientes a un bucle básico del protocolo:

$$P_1 \stackrel{s_1}{\Longrightarrow}_{p_1} P_1 \quad s_1 = \langle msg\, \lambda\, \lambda''\, \lambda' \rangle \;\text{ y }\; p_1 = p$$
$$P_1 \stackrel{s_2}{\Longrightarrow}_{p_2} P_1 \quad s_1 = \langle msg\, msg'\, out\, ack'\, \lambda' \rangle \;\text{ y }\; p_2 = p(1-p)$$
$$P_1 \stackrel{s_3}{\Longrightarrow}_{p_3} P \quad s_1 = \langle msg\, msg'\, out\, ack'\, ack \rangle \;\text{ y }\; p_3 = (1-p)^2$$

Entonces, denotaremos con $f(n)$ la probabilidad de completar el ciclo tras n bucles fallidos, o lo que es lo mismo tras $n+1$ bucles básicos. De las probabilidades de las computaciones previas, este valor puede ser calculado como sigue:

$$f(n) = (p + p(1-p))^n (1-p)^2 = (1-p)^2 p^n (2-p)^n$$

Entonces la probabilidad de finalizar el ciclo en tiempo finito viene dada por la expresión

$$\sum_{i=0}^{\infty} f(i) = (1-p)^2 \sum_{i=0}^{\infty} (2p-p^2)^i = \frac{(1-p)^2}{1-(2p-p^2)} = \frac{(1-p)^2}{(1-p)^2} = 1$$

Esto prueba que el protocolo es seguro, puesto que la probabilidad de tener una transmisión correcta es 1.

El tiempo medio de emisión puede ser calculado por medio del número de emisiones realizadas hasta que un reconocimiento se reciba por el transmisor. Entonces, el tiempo medio de emisión (μ) viene dado por

$$\mu = \sum_{i=0}^{\infty} i\, f(i) = (1-p)^2 \sum_{i=0}^{\infty} i(2p-p^2)^i = (1-p)^2 \frac{2p-p^2}{(1-(2p-p^2))^2} =$$
$$(1-p)^2 \frac{p(2-p)}{(1-p)^4} = \frac{p(2-p)}{(1-p)^2}$$

6 Conclusiones y Trabajos Futuros

Hemos presentado una semántica operacional simple para un lenguaje que permite el modelado de sistemas concurrentes con comportamiento probabilístico. La base del sistema es el clásico CSP, en el que intentamos mantener el significado usual de sus operadores.

Este trabajo constituye sólo un primer paso de nuestra labor en el campo, y en el mismo no se ha conseguido cubrir el operador de ocultamiento, tema en el que nos encontramos trabajando en la actualidad. Para ello, además de controlar en cada paso la acción que se ejecuta, será necesario conocer además el contexto en que dicha acción es ejecutada, esto es, el conjuntos de acciones posibles de ejecutar en ese momento, lo que denominaremos el *estado* del proceso en ese instante. Por lo que una transición será

$$P \xrightarrow{a,A}_p P'$$

que interpretamos como: El proceso P alcanza el estado A, y tras ejecutar la acción a pasa a comportarse como Q con probabilidad p.

Estamos también trabajando en una semántica denotacional del lenguaje, que puede caracterizarse por medio de una semántica de pruebas, así como en una axiomatización completa en el caso finito, de la misma. Todo lo cual formará parte de la Tesis Doctoral del primero de los autores, de próxima finalización.

Si bien los desarrollos anunciados en el parrafo anterior son de una complejidad técnica bastante mayor a lo que se presenta en el presente trabajo, el mismo tiene para nosotros el interés de haber sido el punto de partida sin el cual nada habría sido posible, reuniendo las ideas intuitivas básicas para nuestro trabajo, y que resultan más fáciles de comprender en un marco técnicamente m'as simple, como el aquí presentado.

Referencias

[1] A.V.Aho, J.D.Ullman, M.Yannakakis. *Modeling communication protocols by automata.* Proc. 20 th. IEEE Symp. on Foundations of Computer Science, 267-273,1979.

[2] Ivan Christoff. *Testing Equivalences and Fully Abstract Models for Probabilistic Processes.*, CONCUR-90, LNCS 458, 1990.

[3] A. Giacalone, C.-C. Jou, S.A. Smolka. *Algebraic reasoning for probabilistic concurrent systems.* Pr. Working Conference on Programming Concepts and Methods. IFIP TC 2, Israel, 1990.

[4] R. van Glabbek, S.A. Smolka, B.U.Steffen, C.M.N.Tofts, *Reactive, generative and stratified models of probabilistic processes.* Proceedings of 5th Annual IEEE Symposium on Logic in Computer Science, 130-141, Philadelphia, PA, 1990.

[5] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International 1985.

[6] C.-C. Jou, S.A. Smolka. *Equivalences, Congruences and Complete Axiomatizations for Probabilistic Processes.* CONCUR-90, LNCS 458, 1990.

[7] K.G.Larsen, A.Skou *Bisimulation through probabilistic testing.* Proceedings of 16th Annual ACM Symposium on Principles of Programming Languages, 1989.

[8] C. Miguel. *State of the Art on Timed and Probabilistic Models.* Internal Report DIT - Univ. Politécnica de Madrid, 1991.

[9] R. Milner. *Communication and Concurrency.* Prentice-Hall International, 1985.

[10] G.D. Plotkin. *A structural approach to operational semantics.* Technical Report DAIMI FN-19, Aarhus University. 1981.

[11] W. Pugh. *Skip Lists: A Probabilistic Alternative to Balanced Trees* Communications of the ACM. June, 1990.

Derivación y Verificación de Algoritmos en Control Numérico a partir de una Interfaz Gráfica

Ing. Armando De Giusti[1]
Lic. Laura Lanzarini[2]
Lic. Marcelo Naiouf[3]
Sr. Pedro D'Argenio[4]

Laboratorio de Investigación y Desarrollo en Informática.[5]
Departamento de Informática. Facultad de Ciencias Exactas.
Universidad Nacional de La Plata.

Resumen

Se presenta el desarrollo de un sistema gráfico interactivo que permite derivar y verificar un programa en control numérico para máquinas herramienta a partir de una interfaz gráfica, basada en el diseño mismo de la superficie a trabajar realizado con un sistema de CAD.

Se exponen algunos resultados obtenidos en la primera etapa de experimentación en el Departamento de Ingeniería de una empresa dedicada al desarrollo de matricería para inyección de plástico, analizándose la generalización del sistema a otra gama de equipos industriales.

Palabras Claves:

CAD/CAM, Reconocimientos de Caminos, Compilación, Derivación de código, Simulación

Introducción

En el ámbito industrial, y en particular en la fabricación de matrices destinadas a procesos, es muy habitual la utilización de sistemas de CAD para el diseño y ajuste de herramentales.

A su vez los procesos de elaboración en matricería incluyen normalmente la utilización de máquinas programables para el fresado, cortado o pulido de las matrices. Estos sistemas de

[1]Investigador Principal del CONICET. Profesor Titular Dpto. Informática, Facultad de Ciencias Exactas, Univ. Nacional de La Plata.
[2]Profesor Adjunto Dpto. Informática, Facultad de Ciencias Exactas, Univ. Nacional de La Plata.
[3]Jefe de Trabajos Prácticos Dpto. Informática, Facultad de Ciencias Exactas, Univ. Nacional de La Plata.
[4]Alumno avanzado de la Licenciatura en Informática.Becario LIDI.
[5]50 y 115 1er. Piso. La Plata (1900). Bs. As. Argentina
Tel.: +54 21 42738
Email: lidi@cespivm2.bitnet - lidi@cespivm2.unlp.edu.ar

CAM son habitualmente encapsulados y utilizan un lenguaje dedicado muy limitado como los lenguajes para control numérico, o bien una interfaz gráfica reducida.

Por ello existe un "gap" entre el diseño gráfico y el control directo de la máquina herramienta, que normalmente debe resolverse con sistemas de software dedicado de alto costo.

Nuestro objetivo fue a partir del diseño gráfico realizado con un producto estándar (en este caso AutoCAD), derivar automáticamente el programa en control numérico de una gama de máquinas herramientas y, por otro lado, simular el trabajo de la máquina según un programa dado, a fin de verificarlo "off-line". Inicialmente, el desarrollo se centró en las fresadoras y mandrinadoras MAHO, bajo un mando en control numérico modelo CNC 432, pero, posteriormente, se observó la alternativa de extender el sistema para trabajar sobre cualquier tipo de control numérico a partir de un código genérico.

La principal ventaja de este enfoque es el aprovechamiento integral de la potencia de un sistema de CAD muy probado, como el AutoCAD, respecto del cual existe normalmente un conocimiento de parte del usuario. Esto influye fundamentalmente en la disminución sustancial del costo del sistema ya que el ambiente gráfico para el diseño de piezas es obviado. Otra gran ventaja es la posibilidad de trabajar con una variedad de códigos de control numérico.

Ambiente Gráfico Interactivo para el Desarrollo de Algoritmos en Control Numérico: AGIDA-CN. Ver 2.0

En el ambiente industrial normalmente la verificación del correcto funcionamiento de un programa en control numérico se realiza sobre la máquina herramienta, ya sea trabajando sobre la pieza o bien en un "modo de prueba" a fin de analizar recorridos. En consecuencia, los errores de programacion se corrigen por un metodo de "prueba y error" que eventualmente puede redundar en daños de la pieza y que requiere una fuerte interacción con el especialista de diseño.

Dado que esto representa un costo significativo en tiempo y dinero, sobre todo en matricería que fue nuestro punto de referencia, es muy conveniente disponer de un método menos riesgoso de verificación, ejecutable "off-line".

El AGIDA-CN integra todo el proceso de desarrollo de un programa en control numérico. Se compone de dos subsistemas: un derivador de programas y un simulador de trayectorias.

El subsistema de derivación de programas establece un entorno que permite la obtención del código en forma automática a partir de una trayectoria graficada en un ambiente dedicado, o bien en forma asistida sobre el diseño de la pieza a maquinar.

El subsistema de simulación de trayectorias presenta un ambiente que permite visualizar el recorrido a realizar por la herramienta, incluyendo un conjunto de tests de restricciones propias (ej: no exceder determinadas coordenadas absolutas).

El AGIDA-CN está desarrollado utilizando los lenguajes Pascal y C. En lo que refiere al manejo del ambiente AutoCAD, éste fue modificado utilizando el lenguaje AutoLISP, provisto por el mismo producto.

En cuanto al hardware requerido por el ambiente, sólo es necesario una PC-AT con coprocesador y monitor VGA.

El Subsistema de Derivación de Código

El punto de partida para la derivación del código es un archivo en formato DXF (Drawing Interchange File) obtenido desde un gráfico realizado con una herramienta de CAD. En particular hemos trabajado a partir de AutoCAD como ambiente de diseño al cual se le modificaron los menúes adecuándolo a los requerimientos de nuestra aplicación.

El archivo DXF es compilado a un conjunto de entidades internas que permiten el manejo de la gráfica tales como lineas, arcos, círculos o entidades más complejas como agujeros, punto de partida de la herramienta, comienzo de trabajo de corte, etc.

Este subsistema tiene dos partes claramente definidas. Una es la generación automática del recorrido de la herramienta y otra es la obtención de tal recorrido en forma asistida.

La generación automática se realiza a partir del dibujo de una trayectoria. Tal trayectoria debe ser contínua y el punto de partida de la herramienta está destacado por una entidad especial. El sistema obtiene un conjunto de recorridos posibles que son mostrados al usuario, quien elegirá a su conveniencia. Para completar la información necesaria para concluir en el código en control numérico se solicita al usuario datos adicionales tales como la velocidad de corte, la velocidad de avance, el tipo de herramienta, el tipo de aproximación de la herramienta, etc. Dichos datos serán constantes durante todo el recorrido.

Este tipo de obtención del programa es razonable utilizarlo cuando una única herramienta está involucrada en el maquinado de la pieza y además el corte es contínuo sobre la trayectoria.

La generación asistida puede trabajar directamente sobre el diseño de la pieza a maquinar. El usuario establece el recorrido de la herramienta seleccionando puntos dentro del dibujo y estableciendo si el movimiento realizado es de corte o de posicionamiento. En cualquier momento de la descripción de la trayectoria el usuario puede elegir cambiar la herramienta, la velocidad de corte o de avance, la profundidad, la aproximación, etc.

El Subsistema de Simulación de Trayectorias

Este subsistema genera en forma simultánea, un gráfico en planta de la pieza a confeccionar y una indicación de altura o profundidad de la herramienta en cada momento.

Antes de comenzar la gráfica se calculan las dimensiones del dibujo para poder escalarlo correctamente.

Para la simulación de trayectorias se utiliza una base de datos que indica la sintaxis a verificar y la interpretación de cada comando.

A medida que el programa se va ejecutando se construye el contexto corriente dado por:

- Cero del programa
- Punto actual
- Velocidad
- Ultimo comando ejecutado
- Lista de comandos modales

Durante la simulación la pantalla está dividida en cuatro zonas:

- Una zona donde se va construyendo la gráfica.
- Otra conteniendo las últimas ocho líneas ejecutadas del programa en control numérico.
- Una línea o zona de estado para mostrar algunos valores del contexto corriente.
- Area de menú, con los comandos habilitados.

La simulación de la trayectoria es en tiempo real, mostrando la velocidad de desplazamiento de la herramienta. Puede realizarse en forma automática o en modo de "debugging" estableciendo puntos de parada durante la ejecución.

Descripción Funcional del Sistema

El ambiente presentado fue utilizado en forma experimental en el Departamento de Ingeniería de la empresa COZZUOL S.A. (una de las más importantes empresas de la industria del plástico en la Argentina) obteniéndose resultados alentadores.

Esto nos motivó a analizar la generalización del sistema a otra gama de equipos industriales.

La idea fue desarrollar un lenguaje "genérico" intermedio, de manera tal que el subsistema de derivación, antes de obtener el programa en control numérico para una herramienta determinada, obtenga una descripción en dicho lenguaje. En forma análoga, el

subsistema de simulación toma el programa en control numérico y obtiene la descripción genérica.

Para esto se pensó en tener una Base de Datos de traductores ("Cross-compilers") desde y hacia ese código genérico de manera tal de obtener código ejecutable para cada máquina específica.

El siguiente esquema muestra claramente la arquitectura básica del sistema:

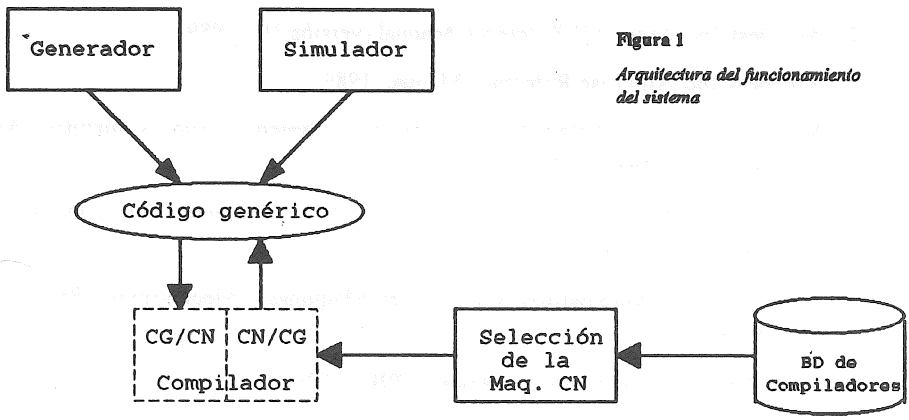


Figura 1

Arquitectura del funcionamiento del sistema

El compilador se instancia a partir de la selección de la máquina de control numérico.

El código intermedio es, en realidad, una estructura de datos definida por una lista de comandos. Los comandos identifican, por un lado, entidades a maquinar como rectas, arcos, círculos, agujeros, y por otro, datos referentes al corte como herramienta, velocidad de corte y de avance, aproximación de la herramienta, etc.

Conclusiones

Se ha desarrollado un sistema de CAM experimental, a partir del diseño gráfico en AutoCAD y se lo ha aplicado a una gama muy utilizada de máquinas herramientas.

No se trata de un producto final, sino de una línea de trabajo en CAD/CAM que representa un área de alto valor agregado en Informática. Esta línea experimental tiene varios temas de interés dentro del LIDI, que se engloban en lo que genéricamente se denomina "Computer Integrated Manufacturing" (CIM).

Agradecimiento

Los autores agradecen al personal del Dpto. Ingeniería de COZZUOL S.A. por su apoyo en el desarrollo y experimentación de este sistema.

Bibliografía

[1] A. V. Aho, R. Sethi, J. D. Ullman. Compilers, Principles, Techniques and Tools

[2] Autodesk Inc. AutoCAD Reference Manual (versión 10). 1989.

[3] Autodesk Inc. AutoLisp Reference Manual. 1989.

[4] M. P. Groover. Automation, Production Systems, and Computer Aided Manufacturing. Prentice Hall. 1980.

[5] W. P. Groover, E. W. Zimmers. CAD/CAM: Computer-Aided Design and Manufacturing. Prentice Hall. 1984.

[6] Lopez Navarro. Automatismo y Control en Máquinas y Mecanismos. Ed. Gustavo Gili. 1975.

[7] E. de Paula Ferreira. Robótica Básica. V EBAI. 1991.

[8] C. L. Tozzi, L. P. Magalhães. Sistemas Gráficos: Técnicas de Modelagem, Desenho, Pac. V EBAI. 1991.

INTEGRATING GRAPHIC SOFTWARE - THE MAKING OF "M.A.T.E"

Sílvia Delgado Olabarriaga
Ana Elisa Schmidt
Carla Maria Dal Sasso Freitas
Manuel Menezes de Oliveira Neto
Marcos Eduardo Nascimento
Rui Bastos
Alexandre Casacurta
Anatólio Laschuk
João Batista de Oliveira
Marcos Kordyas Dossa
Robson Rodrigues Lemos
Soraia Raupp Musse

Universidade Federal do Rio Grande do Sul
Instituto de Informática
Caixa Postal 15064, CEP 91501-900, Porto Alegre, RS, Brazil
e-mail: informatica@inf.ufrgs.br

ABSTRACT: This work describes an experience developed by the Computer Science Graduate Program graphics group at the Federal University of Rio Grande do Sul. The goal was to make a short video showing computer animation completely generated with local software. The video "M.A.T.E" has been produced as a result of this effort. All steps in the production of the video are described: script and storyboard design, objects modeling, motion planning, rendering, sound and final production. Some comments about this experience are presented in the conclusions.

KEYWORDS: modeled animation, computer graphics.

HISTORICAL REASONS

For a long time computer animation has been a very popular subject among computer graphics students and professors of our department. Some simple modeled animation systems have been developed, but people could never see the results on film or video because our laboratories didn't have the appropriate equipment to record computer generated images into these media.

In December'90 the group contacted the computer graphics department of a local TV company (RBS-TV) which has a frame-buffer connected to a U-Matic video

recorder. They agreed to lend us the equipment for an experimental computer animation video and then M.A.T.E. project has started.

SCRIPT DESIGN

The script has been created by the whole group (about 12 people) and this was not an easy task. There were many constraints, some internal, other external, that influenced the choices for a story: we could only use locally developed software (from modeling to animation and rendering); the video should show the "state-of-the-art" of the research work under development; we should do something that local animation houses couldn't (because of the technique itself or cost); the video should be very short.

The discussions lasted more than one month, but finally converged into this story about a chess game between gray and red pieces. Initially two moves are shown (red tower and gray king), while the observer flies around the scene. Then an yellow sphere falls over the chessboard, stops in front of the gray king and turns into a red queen, which creates a checkmate situation on him. A trapdoor under the king suddenly opens and he falls from the chessboard. The story ends when the king passes between two letters "F" and "M" forming the word "FIM" ("end" in Portuguese). The whole sequence lasts 13 seconds. Some frames can be seen on fig. 1.

OPENING

The video opening shows a spiral galaxy of 35000 light reflective particles contracting into a sphere, forming the Computer Science Department logo. The animation was generated through SISPA [1], a particle system specially developed for this project.

OBJECT MODELING

The objects were modeled by two different programs, BEZIER4D [7] and SIHMOS [2], that allow the generation of 3D polygon meshes. The chessboard and trapdoor

models were plane Bézier patches (BEZIER4D); the sphere (rotational sweeping) and letters (translational sweeping) were created by SIHMOS; the chess pieces were Bézier surfaces created through rotational sweeping (BEZIER4D).

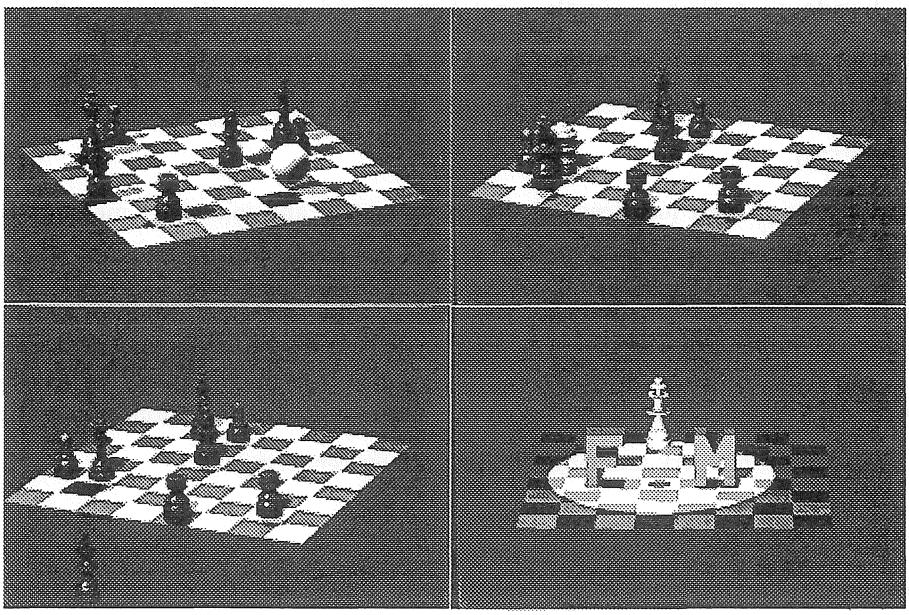


Fig. 1 - M.A.T.E. keyframes

ANIMATION

The camera (observer) and some objects called "actors" (chess pieces and trapdoor) move; the sphere both moves and changes its shape and color.

The camera flies around the chessboard in a spiral movement, slowly going up and always looking at the chessboard center. Key-frame positions for the camera were calculated using a curve-fit program [5] and interpolated by PREVIEW [9].

The pieces move in a straight line from the initial to the final square on the chessboard. The positions were calculated through linear interpolation of key-frames, by PREVIEW.

Sphere's motion simulates a projectile using physically based laws. The intermediate positions were calculated by PROJÉTIL [4]. This actor also changes

its color and shape from an yellow ball into a red queen; the intermediate shapes and colors were calculated by AIF [8].

The trapdoor's motion is simply a rotation around one of its edges and was calculated by PREVIEW by means of key-frame interpolation.

The gray king's final motion is a physically based free fall that was also generated by PROJÉTIL.

Each animation part has been calculated separately and joined by means of a script, which described actor's actions and the timing (frame numbers). This action could be informed only for some key-frames (and then interpolated by PREVIEW) or directly specified for every frame (in files). PREVIEW was the program in charge of synchronizing every actor's and camera motion, generating the description of every animation frame for later rendering.

PENCIL TEST AND RENDERING

The complete animation was first tested on the PREVIEW system [9]. Wireframe images of the scene were quickly seen, and the script was changed to accomplish animator's ideas. When the motion was considered ready, the description of each frame in the sequence has been output to files that were used by the renderer. Each set of files described one frame.

The final images included shadows and Phong shading, and were generated by McSHADOW [6].

SOFTWARE TOOLS

Some software tools already existed and had to be adapted, others were under development as prototypes of ideas for graduate research, and a few were specially developed to accomplish this project. Fig. 2 shows all programs involved, which are shortly described below:

SISPA: particle system simulator [1].

SIHMOS: solid modeler for sweep and parametric representations [2]. The output is a polygon mesh.

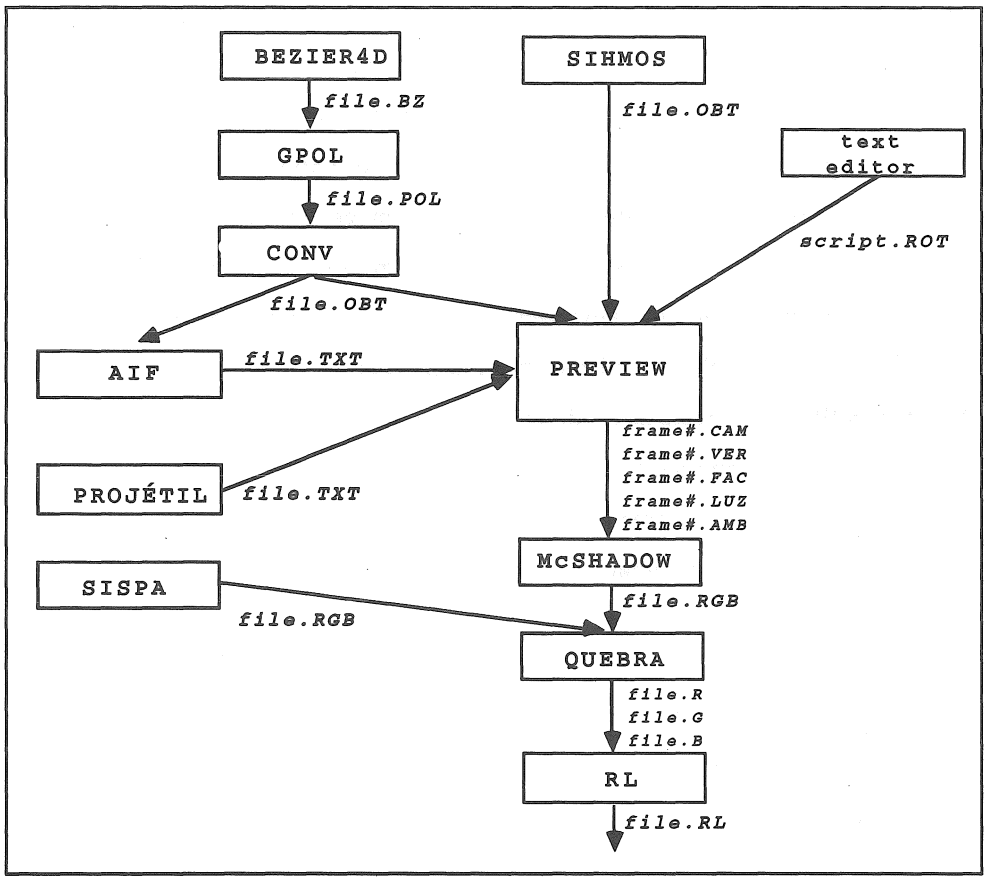


Fig.2 - Software Tools and Files

BEZIER4D: interactive Bézier surface editor and animator [7].

GPOL: converts descriptions of Bézier surfaces into polygons.

CONV: converts polygon files into the local file format.

PROJÉTIL: calculates motion based on inverse kinematics, considering gravity, air resistance and collision with the floor [4]. The motion positions are stored in text files.

AIF: automatically interpolates two given Bézier surfaces [8]. The sequence of shapes are generated into numbered files and it shows a gradual and continuous metamorphosis.

PREVIEW: 3D modeled animation system based on key-frame linear and Bézier interpolation with velocity control through direct kinematic. It reads a script and calculates scene parameters at every frame (camera position and interest, lens, actors transformations, color and shape). This system also displays wireframe images that allow the animator to foresee the result of combined motion. Fig.3 shows a script example showing keyframe interpolation (camera), physically based motion (sphere) and metamorphosis (queen).

McSHADOW: photo-realistic renderer [6]. Simulates colored light sources, Phong shading and shadows. The output is a true-color image file.

QUEBRA: separates red, green and blue image information into three files.

RL: converts local image file into CUBICOMP format (for recording on video).

```
INICIO
NRO_QUADROS= 10                          number of frames

CAMERA
QI= 1 PI= 0 0 0                          camera motion
QF= 10 PF= 5 5 5                         from (0,0,0) to (5,5,5)
FIMCAMERA

ATOR sphere.obt 1
QI= 1 QF= 10 LISTA_POS= positions.txt    sphere's positions have been
FIMATOR                                  calculated by PROJÉTIL

ATOR queen.obt 2
QI= 1 QF= 10 LISTA_FORMA= shapes.txt     queen's shapes have been
FIMATOR                                  calculated by AIF

FIM
```

Fig. 3 - Example of script

The communication among the programs was done through files, usually text-files. Sometimes it was necessary to develop special "filters" to adapt existing file

formats to locally defined ones. The file formats involved in this project are listed below:

.BZ, .POL:	control and polygon mesh of Bézier surfaces.
.OBT:	local "standard" for 3D object description and storage.
.ROT:	text file containing the animation script (see fig.3).
.CAM:	camera specification at a certain frame.
.FAC, .VER:	polygon mesh and vertex information for all objects at a certain frame.
.LUZ, .AMB:	source light and special data for the renderer.
.RGB, .R, .G, .B:	final image (true-color).
.RL:	image in CUBICOMP format, ready for recording on video.

HARDWARE

Three different kinds of computers were used: IBM-PC compatible machines, for object modeling (SIHMOS and BEZIER4D), physically based motion calculation (PROJÉTIL), video opening image generation (SISPA) and image file preparation for recording (QUEBRA and RL); PROCEDA workstation (MC68000 based Brazilian microcomputer), for pencil test (PREVIEW); Sun workstations, for rendering (McSHADOW).

IBM-PC computers could exchange files both with Sun (through NFS) and with Proceda workstations (by serial communication). Image recording was done on a CUBICOMP frame-buffer attached to an IBM-286 compatible. Images were taken from our lab to the frame-buffer in floppy disks.

SOUNDTRACK

The main soundtrack is part of the Carl Orff's "Carmina Burana". The sound effects were specially developed for the required timing on a CASIO CZ-3000 synthesizer. This equipment have been attached to an IBM-PC compatible computer via parallel interface and controlled by program. Both software and hardware were locally developed [3].

VIDEO PRODUCTION

The images (opening + chess game) were recorded on U-Matic video at the TV station. Sound and titling were appended using their regular equipment, with a great help from people from the TV.

RESULTS

Some comments should be done about the video itself: it is too short, and people can hardly see what's happening at the story (the time limit imposed by ourselves was too small); the colors used could have been more carefully chosen (we had no true-color monitor at that time to foresee the results); the camera moves at the same time as the actors, so one sometimes doesn't realize all motion (we couldn't see the animation in real-time before recording).

Comments on software integration must include an analysis about PREVIEW. This tool has been used not only in this project, but also as a quick debugger for all animation work done by the group. Nowadays a new version is under development in Sun workstations, with window interface and a revised script language. Also, the output file format has been changed to accomplish new renderers.

CONCLUSIONS

In spite of these comments, the experience has proven to be very important to our group: people who worked with animation could finally see the results of their work; the final result (the video itself) is something we can use to show other people what we are able to do at this moment; many graphics programs were used together and this was unusual in our group in the past; about 12 people worked together and had to learn how to interact with each other, not only from the software point of view.

Before M.A.T.E has been developed, everyone used to build his own modeler, render and so on. This means that people almost started from scratch and

couldn't get very far. Nowadays we have internal file formats, libraries and generic tools which improve the reusability of locally developed software.

THANKS

To Rubem Pechansky, Bayard da Silva Christ, Fábio Rebello, Yara Lemr, Márcio Pinho, Flávio Cauduro and RBS-TV.

BIBLIOGRAPHY

[1] R. Bastos & J. B. Oliveira. *Animação de um sistema de partículas refletoras de luz.* 1991. (Personal Notes).

[2] A. Casacurta. *SIHMOS: Sistema Híbrido de Modelagem de Sólidos.* Porto Alegre, CPGCC-UFRGS, 1991. (MSc).

[3] M. Dossa. *Controle do CZ-3000 por microcomputador.* 1991. Personal notes.

[4] R. Lemos. *Controle de movimentos em animação: teoria e aplicação experimental.* Porto Alegre, CPGCC-UFRGS, 1992. (MSc).

[5] Y. Lemr. *Análise de deformações em superfícies Bézier bicúbicas.* Porto Alegre, CPGCC-UFRGS, 1991 (TI 223).

[6] M. Nascimento. *Um estudo de algoritmos para geração de sombras projetadas na síntese de imagens foto-realísticas.* Porto Alegre, CPGCC-UFRGS, 1992. (MSc).

[7] M. Oliveira Neto & R. Bastos. *BEZIER4D - ambiente para modelagem, visualização e animação tridimensional.* In: SIBGRAPI'91, São Paulo, 1991.

[8] M. Oliveira Neto. *Algoritmo Interpolador de Formas entre objetos modelados por superfícies spline.* Porto Alegre, CPGCC-UFRGS, 1992. (MSc)

[9] A. Schmidt & S. Musse. *PREVIEW - sistema de animação.* Porto Alegre, CPGCC-UFRGS, 1991. (Research Project 184).

GEOHPAR - RESEÑA GEOGRÁFICA E HISTÓRICA DEL PARAGUAY

Hugo Díaz Lavigne
Alberto Grijalba Cabrera

Universidad Católica "Nuestra Señora de la Asunción"
Departamento de Análisis de Sistemas
CC 1718 - Asunción - Paraguay

I. Introducción

El problema del tratamiento de la información, afecta con mucha fuerza, especialmente a los países subdesarrollados, los cuales no cuentan con una política definida y clara, ni con recursos humanos y técnicos suficientes para enfrentar el problema.

En el Paraguay este problema se presenta en grado superlativo. Existen bastantes inconvenientes para el registro y la obtención de información documental. Ante una mínima exigencia, no se sabe exactamente dónde recurrir, ya que ciertas instituciones que deberían realizar este servicio, no están preparadas para ello o simplemente no tienen forma de centralizar o relacionar la información.

Esto aunque es bastante grave, se presenta en forma frecuente en nuestro país, en lo que se refiere a datos geográficos, históricos, turísticos y similares, razón por la cual las informaciones de carácter público que se pueden conseguir, son en muchos casos, nulas o requieren de una recopilación manual.

Otro ámbito en el cual se presenta el problema es en instituciones educativas, donde existen problemas similares. La alternativa es recurrir únicamente a profesores que conozcan un tema específico, ante la carencia de textos, desaprovechándose constantemente los trabajos de investigación realizados por alumnos o profesores, generalmente por inconvenientes técnicos referentes a la organización, almacenamiento y relacionamiento de la información.

GEOHPAR es un *Sistema de Información Geográfico*, que tiene como objetivos la integración gráfica y documental de los datos, con el objeto de dar soluciones reales al problema de tratamiento y visualización de la información.

Es importante significar que **GEOHPAR** es un Sistema de Computación recientemente acabado, que representa el fruto de un grupo de trabajo, formado por un profesor de la carrera de Análisis de Sistemas de Universidad Católica de Asunción, recientemente recibido, y un alumno de la carrera de Ingeniería de Sistemas de la misma institución académica.

II. Cómo nace la idea ?

Geohpar es un proyecto cuyo inicio se remontan al mes de julio de 1989.

A inicio de dicho año, se produce un cambio de gobierno en el país, luego de 35 años. Esto hace que el hombre paraguayo en general tenga real conciencia en que al país lo debemos construir entre todos. Este factor ayuda notablemente a hacer efectiva la idea.

La idea surgió de un Profesor del área de Informática de la Universidad Católica de Asunción (Hugo Díaz Lavigne), quien vende la idea a un grupo de trabajo, y juntos realizan dicho proyecto, hasta su finalización, en octubre de 1991.

A partir de allí, se comenzó a trazar los planes para poder armar un proyecto sin precedentes en el país, que persiga 2 objetivos; presentar una tesis de grado con una calidad total, que sea apreciable no sólo en nuestro país, sino también en otras latitudes; y servir para ayudar a diferentes sectores de la sociedad paraguaya, con el producto.

El primer objetivo se ha cumplido en parte, ya que se ha presentado la tesis y ha obtenido la nota 5 FELICITADO.

III. Qué es GEOHPAR ?

GEOHPAR es un Sistema de Información Geográfico, ideado y realizado íntegramente en el Paraguay, que tiene como objetivos la integración gráfica y documental, con el objeto poder tener al país en la mano, en el momento de realizar las consultas en el Sistema.

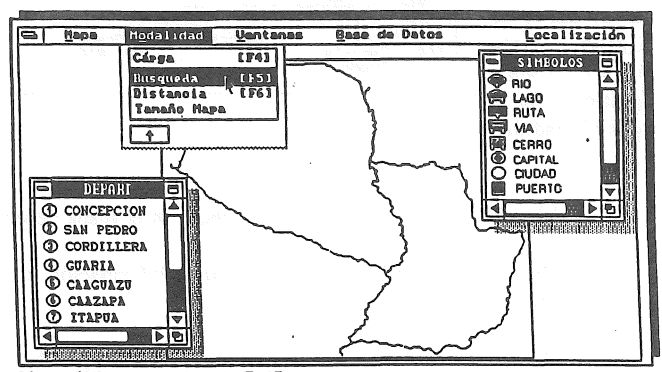


Figura 1- Ventanas y Menús Gráficos

Comprende una Reseña Geográfica e Histórica del Paraguay, y trata de una apreciación Gráfica integral de nuestro país, unida a un potente Banco de Datos, que posibilita la visualización y el acceso a diferentes tipos de información.

GEOHPAR es una herramienta adaptable a varios propósitos, ya que uno puede cambiar totalmente la información que inicialmente existe, con el fin de adaptarlo a una situación específica.

IV. Cuáles son sus características principales ?

GEOHPAR incorpora dos elementos bastante interesantes, hasta ahora ausentes en este tipo de trabajos :

1) La asociación gráfica-documental de los componentes del sistema (el Mapa en sí y el Banco de Datos relacionado), de manera que cualquier consulta o aprendizaje resulte en una agradable tarea, y que a su vez incentive la investigación, y ;

2) La posibilidad de relacionamiento de los datos, asociándolos según el contenido de la información inmersa en el Banco de Datos.

Inicialmente el Sistema presenta un compendio de datos referentes al Paraguay, en un mapa, cuyas características de presentación está apoyada en un banco de datos. Esta interfase gráfica, da la posibilidad de:

PARAGUAY RESEÑA GEOGRAFICA E HISTORICA Fecha:31/12/91
TIPO : TODOS DEPARTAMENTO : 4 GUAIRA
NOMBRE :

No.	TIPO	DEPA	N O M B R E DEL PUNTO	FECHA FUNDA	CANT. HAB.	EXT.	DIST ASUN	COORDENADAS X	COORDENADAS Y	POSI	A I	A F
1	CI	4	BORJA	01/01/	12944	89		214000	262000	DC	0	0
2	CI	4	CNEL. MARTINEZ	/ /	6440	74		208300	254200	DC	0	0
3	CI	4	COL. INDEPENDENCIA	01/01/	46486	147		224000	252000	DC	0	0
4	CI	4	FELIX PEREZ CARDOZO	/ /	4816	55		217000	253500	DC	0	0
5	CI	4	GRAL. EUGENIO A. GA	/ /		67		224000	262000	IC	0	0
6	DE	4	GUAIRA	/ /	181936	3022		225000	257000	DC	0	0
7	TU	4	GUAIRA	/ /							0	0
8	CI	4	ITAPE	01/01/		70		209000	257000	DC	0	0
9	CI	4	ITURBE	/ /	8927	90		216000	264000	DC	0	0
10	CI	4	JOSE FASSARDI	/ /		46		226000	262500	DC	0	0
11	CI	4	MAURICIO J. TROCHE	01/01/		44		225000	248500	DC	0	0
12	CI	4	MBOCAYATY	/ /							0	0
13	CI	4	MBOCAYATY DEL YHAGU	/ /		63	162	216000	252000	DC	0	0

[Nº] Línea [I] Inserta [G] Graba [F] Fin
[ESC] Cancelar [PG SIG] Esc: Cancelar
F1: Ayuda

Figura 2- Ambiente de Base de Datos

- Visualizar los datos con que se cuenta en el mapa de nuestro país.
- Ofrecer la posibilidad de ir agregando información, de acuerdo con las necesidades de cada persona o institución.
- Realizar acercamientos sucesivos en el mapa, hasta llegar al nivel de un lugar cualquiera, en alguna ciudad.
- Poder relacionar lo que gráficamente se ve, con una buena fuente de información que está contenida en la Base de Datos.
- Acumular datos de cualquier índole en el Banco de Datos, además de las que aparecen en el mapa, con el fin de realizar consultas referentes a cualquier tópico, sean visualizables o no gráficamente.

V. *Análisis Técnico del Producto*

GEOHPAR es un programa que puede ser utilizado en un computador personal IBM o compatible.

Consta de aproximadamente 20.00 líneas de código fuente, desarrollados íntegramente para el proyecto.

Utiliza una interfase gráfica construída especialmente para este proyecto, similar al Windows, pero que no requiere ningun programa adicional.

Los módulos desarrollados pueden diferenciarse en los siguientes grupos:

- Módulos Gráficos: Compilador Turbo C 2.0
- Módulos de Base de Datos Compilador Clipper 5.01
- Integración de Ambientes A través de los archivos de Datos

A continuación se desarrollarán algunas de las características técnicas del producto, que hacen a esta investigación, única en su género en el Paraguay.

A. Módulos Gráficos:

Es la parte del sistema en la cual se ha realizado el 80 % de la investigación.

Se ha realizado la programación íntegra de los módulos. Esto se ha hecho así, porque el trabajo constituía un trabajo de Tesis, y en el Paraguay hay muy poco desarrollo para el ambiente gráfico.

Los módulos claves en este desarrollo son:

1. La interfase para el manejo del ratón.
2. El manejo de cursor o ratón simultáneo.
3. El manejo de las pantallas gráficas, cualquiera sea su resolución.
4. La modalidad de interacción con los menúes del Sistema, utilizando la técnica de multiventanas y de ventanas solapadas o activas.

B. Módulos de Base de Datos

La posibilidad de aprovechamiento de las posibilidades de manejo de información que brinda una Base de Datos, ha hecho de esta, una poderosa herramienta para el tratamiento de la información documental.

Los módulos claves de este desarrollo son:

1. Creación de un Diccionario de Datos mínimo
2. Utilización de Técnicas de optimización del rendimiento
3. Módulo de búsqueda por contenido o descriptores.

C. **Integración de Ambientes**

Este punto es uno de los cuales más se investigó, ya que existían varias alternativas al respecto.
Finalmente, se adoptó el criterio de trabajar en ambientes integrados, sólo a través de los archivos de datos, posibilitando cualquier tipo integración de tareas.

VI. Qué elementos se utilizan para resolver los problemas ?

GEOHPAR está formado por los siguientes componentes:

Ambiente Gráfico : Es el momento en el cual se presenta el mapa donde uno puede realizar cualquier actividad con las herramientas gráficas.

Banco de Datos : Constituye la fuente de información. Es el conjunto de los datos almacenados.

Puntos : Son los puntos individuales que aparecen en el mapa. En ellos está contenida la información, las cuales están diferenciadas de la siguiente manera.

Datos Fijos: Es la información fija, característica de cada punto.

Datos Variables: Son los datos propios de la Base de Datos. Están formados a su vez por el texto en sí, y las palabras claves del texto.

Líneas : Son las líneas continuas del mapa. Se utilizan para representar cualquier información que deba ser representada como una raya o conjunto de rayas en el mapa. *Los límites del Paraguay, los ríos, la divisoria de Departamentos, son ejemplos de Líneas.*

Tablas : Se llama así a las demás tablas auxiliares, en las cuales de apoyan las anteriores expuestas.

VII. Requerimiento del Equipo de Computación

GEOHPAR funciona en cualquier computador personal compatible con IBM, desde una XT.

Requiere de alguna de las siguientes tarjetas gráficas: Hércules, EGA o VGA.

Es conveniente tener un ratón, aunque no es excluyente.

VIII. Cómo contactar con GEOHPAR ?

GEOHPAR
Departamento de Análisis de Sistemas
Universidad Católica "Nuestra Señora de la Asunción"
Teléfono : 334-650
Casilla de Correos 1817
Asunción - Paraguay

GEOHPAR
Teléfono : 497-416 Hugo Díaz Lavigne
Casilla de Correos 2006
Asunción - Paraguay

Observación: ***Esperamos que este corto resumen haya podido dar una idea de la envergadura del trabajo realizado, ya que en él se han invertido dos años y medio de esfuerzos y dedicación constante. Quedamos a vuestra disposición para cualquier consulta adicional.***

A GOAL MODEL FOR THE EVALUATION OF VIDEOTEX SYSTEMS

E. Domínguez, I. Escario, M. A. Zapata, M. J. Lapeña, A. Miguel

Dept. de Ingeniería Eléctrica e Informática
Facultad de Ciencias. Universidad de Zaragoza
50009 Zaragoza (Spain). E-mail: ccia@cc.unizar.es

Abstract. A model for the evaluation of the user goals that are observed in the user interface of a Videotex System is presented here. This concept-oriented model allows specification of the goals that an application shows, or may show, as well as the system behaviour in response to the user action.

Keywords: User interface, goals, evaluation, Videotex, Ibertex, multimedia databases.

1.- Introduction. A Videotex System is an interactive system which may offer a wide variety of services: access to information, transactions (shopping, booking, etc.), amusement games and so on. In designing this type of systems, the user's preferences should have priority over any system-oriented considerations. That is why the user interface is crucial in this kind of systems.

In recent years numerous contributions (see [6]) have been made to the integration between Software Engineering and Human-Computer Interaction, among which the most notable attempt has been the User Software Engineering Method by Wasserman (see [9] and [10]), but a consistent model oriented to specification, design and evaluation of user interface is still missing. The greatest difficulty is that in the user interface several aspects of a very different nature (control, navigation, information structuring, ergonomy, dialogues, user tasks, etc.) have to considered.

From our point of view a suitable solution might consist of developing a theory that provides the necessary tools to construct concreta models which allow specification of every general aspect of user interface, as well as a suitable methodology for design and evaluation. That is why we are developing a theory that intends to abide by the former conditions. This theory, named Theory of Conceptual Modelling [2], is based on the same basic principles as those used by Kangassalo (see [7]) for the conceptual database design.

The purpose of this paper is to present a model, named Goal Model, which has been adapted to the evaluation of the goals that a Videotex application shows, or may show, by means of the user interface. This model is restricted to the Videotex databases that offer

transactional or information services. An example of a hypothetical Videotex application, which we will call Eli_Bookshop, will be used during the exposition.

Within the Theory of Conceptual Modelling, the following stages must be overcome in order to define a concrete model: 1) Defining the perspective of the model, by means of which the general aspects to consider and the way in which they are observed by the evaluator are established; 2) Establishing the properties of the former aspects that are desired to be analysed; 3) Defining which model allows specification of the aspects that have been observed in an application and to evaluate the properties.

The evaluation of a specific application goes by the following stages: 1) Defining the tools to aid evaluation which, usually, will be schemes of data models; 2) Specifying, within the defined model, the aspects that have been observed; 3) Evaluating the properties and establishing the final conclusions.

In this paper, we are going to limit ourselves to presenting the stages of defining a model.

2.- Our perspective of the model. A Videotex database is oriented to displaying information that is stored in it, or to carrying out a transactional service, decided on by the user. The information it displays to the user (e.g. set of data on a book that Eli_Bookshop has in stock) is a structured set of data with non-null information content, that is to say, information that may provide the user with new knowledge. A transactional service (e.g. the purchase of a book) carries an update of the database due to request by the user who accedes to this service. The name we will give to each piece of information shown or services offered by the application will be *final goals*.

The number of final goals that generally offer a Videotex application is high. This is why the designer is obliged to carry out a detailed analysis of the final goals, grouping them into types and generalizing these groups with a view to providing the user with paths of access which will efficiently lead him to the required final goal. To do this, the application displays various identifiers of different types of information or transaction on the screen from among which the user must choose one. *Options* will be the name we give to each type of information or transaction that the application displays to the user as an aid in access to the final goals and in such a way that its selection causes a change in the state of the application which the user can notice.

Menu is understood as being a set of options displayed on a screen. This set represents a certain level of the classification of the final goals. However, due to the restrictions in the number of lines that Videotex video devices have, on some occasions the number of options that might be needed cannot be displayed. Under these circumstances it is necessary to display the set of options on a series of consecutive screens. It therefore seems convenient to define *menu* as a set of options that the application displays to the user, from among which he may choose one, independently of the number of screens that are needed for their display.

To facilitate the location of the option which matches or may match a user's goal, the application displays the options on the screen with certain attributes (colours, spatial distributions, graphic separators, etc.) which differentiate each other and possibly distributes them into various groups. In this way the user can reduce searching space if he first locates the group where he foresees that the option that matches or approaches its goal is there and, later, carries out the search within the chosen group. These groups, extracted through the differences marked by the display attributes can be abstracted like a tree on which the father node represents the screen. Should the menu be displayed on various screens, the trees extracted from each one can be extended thus forming only one tree, by associating a node to the menu and stating as its sons the nodes that represent each screen on which the options are distributed.

Decision tree is the name we will give a tree built up as we explained in the previous paragraph. In this tree the root always represents a menu and the leaves represent options. Every remaining node of the tree represents a subset of menu options which we will call *submenu*. Next we shall consider that an *application goal* is any menu, submenu or option of an application.

3.- Properties and informal criteria for a good design. For the presentation of the model, we are going to indicate only the properties and criteria that, to our judgement, are the most important.

Firstly, it is necessary for the application goals and the user's goals to match; that is, the goals displayed by an application should match those expected by a user. With this idea in mind, two basic criteria are established which must be fulfiled by an application: *Completeness* - which determines that the user's final goals may be reached with the application, and *minimality* - which establishes that the application must not display unnecessary goals (i.e. goals that the user will never expect to reach).

Secondly, the structuring of the goals as well as the syntaxis and semantics of the identifiers must be correct. Related to this type of properties the following criteria can be established: *Friendliness* - which determines the knowledge of the potential users, and *structural correction* - which establishes that the goals the user reaches on choosing a concrete goal must be those expected.

4.- Goal Model. Depending on the properties and criteria previously established, the evaluator must build a model to specify the aspects he observes in a concrete application and, afterwards, evaluate the degree of fulfilment of the rules or criteria. In this section, we are going to present a Goal Model which suits the aspects and rules that, as an example, were previously described. The basic elements of the model are concepts, where a concept is defined in terms of the Universe of Discourse and is identified by a name. A concept is used in order to represent the abstraction of any observable fact or event.

For example: 1) MENU is the abstraction of a complete set of consecutive screens in

such a way that every one of them contains options and the user can choose one, and only one, from among all the displayed options; 2) ADVANCE is the abstraction of one action or a sequence of actions which take the user from one goal to another without the application presenting a third goal between them. Notice that both concepts provide formalized notions about certain aspects of user interface, which are independent of a concrete application. Nevertheless the former abstracts a possible entity noticeable in user interface while the latter represents a certain beaviour of the user/application system. The concepts of the former type are named *statics* and those of the latter are called *dynamics*.

A mechanism is needed to specify the aspects that are observed in a concrete application. For example, in the application Eli_Bookshop the existence of a screen named "General Menu" is observed, on which the application displays a set of options from among which the user can choose one. This screen can be interpreted as a new concept, abstracted by MENU, which we will call Main_Menu. The meaning (or definition) of Main_Menu could either be the above-mentioned or it could be extended with the enumeration of the options that it displays The Theory of Conceptual Modelling provides a builder named *concept value* which is applied to those concepts that are independent of a concrete application, and which associates to them a concept representing a fact or event, observed in a concrete application, which is abstracted by the given concept. Thus, Main_Menu is a value of the concept MENU.

Sometimes, it is necessary to describe types of specific properties of the static and dynamic concepts with the aim of evaluating more accurately what is observed. To do this, new concepts named attributes are defined and are associated to the concept which is described by them. For example, the attribute APPLICATION_NAME, which is the abstraction of the identifiers with which the application presents a goal, can be associated to the concept MENU.

The attributes are also concepts independent of a concrete application, so the builder concept value can be applied to them. For instance, a value of APPLICATION_NAME is "General Menu" which is the name that the application associates to the value of MENU which we have called Main_Menu.

When any aspect of user interface is to be evaluated, it is necessary to accurately define each and every type of aspect that is to be considered. The specification of these types is performed through the named Support. A *Support* is a set of concepts, independent of the concrete applications, by means of which each and every type of aspects to be considered during the evaluation is represented.

The static and dynamic concepts of the Support are structured as trees in such a way that a concept A is a son of a concept B if, and only if, A is a specialization of B. In this sense, a Support is a series of hierarchical trees of concepts in which leaves make up a set of concepts to be used in specifying the observable, whereas the rest of the concepts represent generalizations of the previous, providing a classification which will help the normalized analysis of user interface. In order to define static and dynamic concepts of a concrete Support,

the Theory of Conceptual Modelling provides the specialization and the generalization of concepts as abstraction mechanisms.

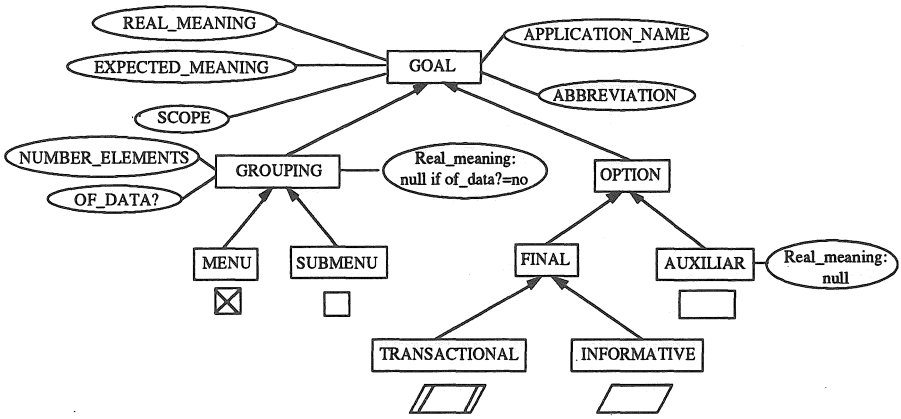


Figure 1.- Goal tree

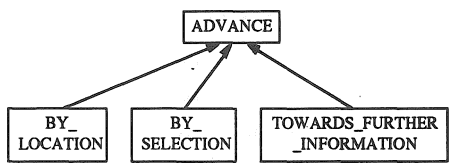


Figure 2.- Advance tree

The Support can also contain attributes. The association of the attribute to the concepts that they describe goes by the following principles: 1) *Inheritance* - it is considered that an attribute associated to a concept is associated to all descendants of this concept, too; 2) *Null value* - attributes able to take a null value are allowed; 3) *Constraint* - when a constraint is imposed on an attribute, this is spread to all descendents of the concept on which this restriction is specified. Other basic principles that Support has to comply with too, are: 1) *Completeness* - every fact or event that the evaluator wants to observe must be abstracted by some concept of the Support; 2) *Classification* - every fact or event that the evaluator wants to observe must be abstracted by one and only one of the terminal concepts of the Support.

The Support associated to the Goal Model consists of a tree of static concepts, named *goal tree* and a tree of dynamic concepts, named advance tree. The goal tree (figure 1) classifies the different types of goals considered in user interface and the advance tree (figure 2) represents the different types of actions. In figures 3 and 4 the definitions of the concepts of the Goal Support are shown.

NAME	MEANING
OPTION	is the abstraction of an entity that the application displays using a name whose meaning is either a possible type of information or modification on the database, or a criterion for access to one of the previous types, and such that its selection produces a change, noticed by the user, in the state of the application.
FINAL OPTION	is an option whose selection by the user produces an access to database which is noticed by the user before reaching the following goal
TRANSACTIONAL	is a final option which carries an updating of the database
INFORMATIVE	is a final option which does not carry an updating of the database
AUXILIARY OPTION	is an option which is not final
GROUPING	is the abstraction of a set af actions represented as a non-terminal node in a particular decision tree
MENU	is a grouping represented as the root in a particular decision tree
SUBMENU	is a grouping not represented as the root in a particular decision tree
GOAL	is the generalization of the concepts OPTION and GROUPING
ADVANCE	is the abstraction of one action or a sequence of actions which take the user from one goal to another without the application presenting a third goal between them
BY_ LOCATION	is an advance defined by a mental action of the user in the menu which he performs in order to locate a goal, provided that this action is represented as an edge of the decision tree of the menu
BY_SELECTION	is an advance defined by a control action which consists of the selection of an auxiliar option
TOWARDS_ FURTHER_ INFORMATION	is an advance defined by several actions, where the first one is the selection of a final option, i.e., it produces an access to the database noticed by the user before reaching the following goal

Figure 3.- Static and dynamic concepts of the Support

NAME	MEANING
APPLICATION_ NAME	is the abstraction of the name given to a goal by the application; it admits null value
ABBREVIATION	is the abstraction of the abbreviation associated to a goal by the analyst to identify it in the diagram and in the tools that aid evaluation
SCOPE	abstracts the fact of whether a goal is accessible to any user (public) or not (private)
EXPECTED_ MEANING	abstracts the meaning that, to the judgement of an analyst, a user would associate to a goal in the moment he identifies it on the screen
REAL_ MEANING	abstracts the meaning of the access to the database associated to a goal; it admits null value
NUMBER_ ELEMENTS	abstracts the number of sons that a grouping has on the decision tree
OF_DATA?	abstracts the fact that at least one son of a grouping represents a value of the database (yes) or none of them does (no)

Figura 4.- Attributes of the Support

In figure 1, shown next to the Support are the graphical describers associated to each of its terminal nodes. These describers are used in connection with another essential component of the model, the Diagram, whose purpose is to represent statically the behaviour of the user interface. Given a concrete application, the *Diagram* is made up of all the values of the concepts specified in the Support which are observed in this application. The graphical representation of the Diagram is carried out by means of a type of graph in which the nodes are values of observed goals along with the values of their attributes and an edge between two nodes represents an advance value between the goals representing these nodes. Every node of the graph will be represented by using the describers corresponding to the concept of which this is a value. In figure 5 we present the Diagram associated to the application Eli_Bookshop, in which, for simplicity, not all values of the attributes have been presented. As it can be observed, this diagram does not fulfil some of the criteria which have been established for a good design. If the stages of evaluating an application are followed, this would be detected.

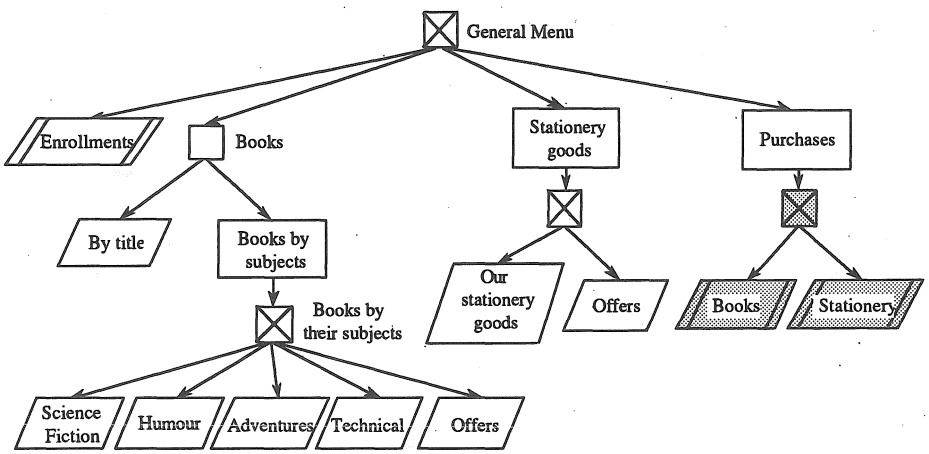


Figure 5.- Diagram of the Eli_Bookshop

5.- Final remarks. The evaluator, depending on his perspective and the particular aspect he wishes to analyse concerning the goals of an application may extend or modify the presented Support. In fact, owing to space limits, this support is a simplification of the one we use in real cases which has been experienced through the evaluation of Ibertex services (Ibertex is the Spanish Videotex Service). Following the Theory of Conceptual Modelling, specific models have also been developed, [3] and [4], for evaluating other aspects of user interface.

Acknowledgements: We wish to thank A. Francés, J. Rubio, A. Cebollada and J. Lloret for their very useful comments during the preparation of this article. We would also like to thank J. L. Rebollo for their invaluable advice on the Ibertex System. This paper has been supported by Telefónica de España S.A. y CICYT (Comisión Interministerial de Ciencia y Tecnología; Project INF89-0879).

References

[1] S.K. Card, T.P. Moran, A. Newell, *The Psycology of Human-Computer Interaction*, Lawrence Erlbaum Assoc., London 1983.

[2] E. Domínguez, I. Escario, M.J. Lapeña, M.A. Zapata, A Theory of Conceptual Modelling, to appear.

[3] E. Domínguez, M A. Zapata, A Model for Human-Computer Interaction, to appear.

[4] E. Domínguez, M. A. Zapata, I. Escario y A. Cebollada, A Control/Navigation Model for the evaluation of Videotex Systems, to appear.

[5] I. J. Elson, Design Trends in Videotex Systems, *Videodisk and Optical Disk* 4(5), 1984, 357-364.

[6] M. Helander (Ed), *Handbook of Human-Computer Interaction*, North Holland, 1990.

[7] H. Kangassalo, Foundations of Conceptual Modelling: A theory construction view, in *Information Modelling and Knowledge Bases*, H. Kangassalo, S. Ohsuga, H. Jaakkola (Ed), IOS Press, 1990, 19-35.

[8] W.A. Simcox, A Framework for the inclusion of Human Factors in the Design of Videotex Systems, *Information Design Journal* 3, 1983, 215-230.

[9] A. I. Wasserman, Developing Interactive Systems with the User Software Engineering Methodology, in *Proceedings Interact-84*, B. Shacked (Ed), vol. 1, 1984, 472-477, North Holland.

[10] A. I. Wasserman, P. A. Pircher, D. T. Shewmake and M. L. Kersten, Developing Interactive Information Systems with the User Software Engineering Methodology, *IEEE Transact. in Soft. Eng.* 12(2), 1986, 326-345.

DIGITIZED SEGMENTS AND STRAIGHT LINES

E. Domínguez°, A. Francés°, A. Márquez*

° Dept. de Ingeniería Eléctrica e Informática
Facultad de Ciencias
Universidad de Zaragoza
50.009 Zaragoza (España)

* Department of Computer Science
Brown University
Providence
Rhode Island 02912-1910 (USA)

Abstract. The goal of this paper is to introduce a code, called stair code, to represent digital segments. We present a new incremental algorithm for drawing segments which is related to this code. We compare it with Bresenham's algorithm and observe that the outputs of these two algorithms give us an appropriate definition of digital segment.

Key words: Digital Geometry, digital representation, encoding, digitization, digital segment, digital straight line.

§1. Introduction. The aim of Digital Geometry is to find the most appropriate data structures and algorithms for solving geometric problems efficiently. Each data structure that encodes a given digital object requires a different algorithm to solve a particular problem. Consequently, to solve a problem, one structure may be better than another depending on its space complexity and the time complexity of its associated algorithm.

The task of finding a suitable code to process digital objects is, therefore, of great importance. The aim of this paper is to present a code for processing segments, called stair code, comparing it with other codes by means of two basic problems. These problems are the drawing of a segment, which will be called DRAWING problem and the question of deciding whether a digital point belongs to a segment which will be called BELONG problem.

The most elemental code for encoding segments is to give two points which are the segment end points. This code, which we will call *minimal code*, has a constant space complexity and it is well known that, using it, Bresenham's algorithm [1] solves the DRAWING problem in optimal time, since it is of the order of the number of pixels to be illuminated. The BELONG problem can also be solved in constant time by using integer divisions and products.

Since the number of pixels that must be illuminated in order to draw a segment on a computer screen is finite, we can interpret the enumeration of these pixels as a code that

we will call *the complete code*. This code is the structure that contains most explicit information on the segment but its storage cost is as great as the number of pixels that make the segment up. The algorithm for the DRAWING problem is trivial using this code; illuminating the pixels contained in the code is enough and its complexity has the same order as the Bresenham's algorithm complexity. The BELONG problem can be solved in constant time using sums and comparisons only if the code is supported by a suitable data structure.

A more accurate structure, which is already classical, for representing digital objects is the chain code [9]. The *chain code* of a segment consists of one point (one of its end points) and a sequence of symbols that represent horizontal and diagonal movements. These movements are those needed for reaching the other end point. The storage cost of this code is, as in the previous case, of the order of the number of pixels which make up the digital segment. Its associated drawing algorithm is also trivial and has the same time complexity as Bresenham's algorithm. However, in the worst case, the algorithm that solves the BELONG problem needs to go through all the code.

In comparison, the minimal code is the best of the above mentioned for solving the DRAWING and BELONG problems because it is the one that needs less space and its associated algorithms are optimal.

For simplicity, we will restrict the problem to drawing and encoding segments with (0,0) and (m,n) as end points, where $m > n > 0$. By s(p,q) we will denote the segment whose end points are p and q, and by $\lfloor x \rfloor$ the greatest integer which is less than the real number x.

§2. Bresenham's algorithm. On a computer screen, any picture can be obtained by illuminating elemental surface areas called pixels. The screen model on which we base this is made up of small squares within an ortogonal grid, which admit two states: black and white. As is shown in figure 1, the display of any linear object (pixels in black) looks locally like a staircase. Only the high resolution of video devices makes the representation appear, to the human eye, to have the same property of linearity as the represented object.

The mathematical model which we use to represent the screen model is an ortogonal grid ($\mathbb{Z}^2$) on which each node (point of integer coordinates) stands for one pixel (figure 2).

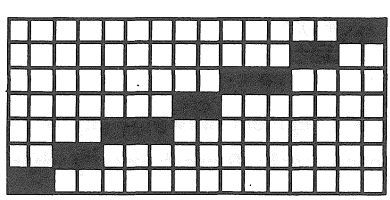


Figure 1. Screen Model

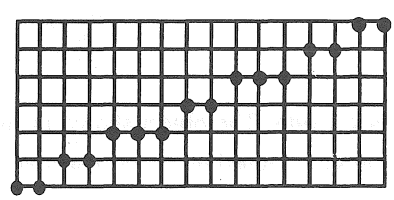


Figure 2. Mathematical Model

With the mathematical model that we have just shown, the problem of drawing a segment on a computer screen was solved by Bresenham calculating the closest integer coordinate points to it.

For each abscissa from 0 to m, Bresenham's algorithm determines, from the pixel P_{i-1} calculated in the previous step, the closest pixel to the segment. As is shown in figure 3, P_{i-1} determines the candidates T_i and S_i, and selects one or another depending on its distance from the actual segment point with this abscissa (being t_i and s_i, respectively). If $t_i \leq s_i$ then T_i is chosen as best approximation (i.e., $P_i = T_i$); otherwise $P_i = S_i$. The effective calculation of these distances requires real arithmetic, but as we only need to determine which is the shorter, we would only need to use a parameter of decision, d_i, which always takes integer values and is proportional to the difference of both distances, $t_i - s_i$. Observe that if, for a given abscissa, both candidates are equidistant from the actual segment point with this abscissa, the algorithm then chooses, by convention, the one with greater ordinate.

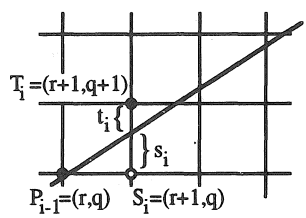


Figure 3. The Bresenham's algorithm chooses, from two candidates, the closer pixel to the segment

The algorithm begins by illuminating the pixel (0,0) and sets the parameter of decision: $d_1 = 2n-m$; then iteratively decides on the following pixel to be drawn and updates the parameter of decision. To be more precise: 1) if $d_i \geq 0$, the pixel T_i is chosen and $d_{i+1} = d_i + 2(n-m)$; 2) if $d_i < 0$ then S_i is chosen and $d_{i+1} = d_i + 2n$. Observe that the asymptotic time complexity of Bresenham's algorithm is O(m).

It can be easily proved that the points which Bresenham's algorithm draws verify the result that is presented in the following proposition. Its proof is straightforward.

Proposition 1. If Bresenham's algorithm calculates the point (p,q) where the input segment is s((0,0),(m,n)) the following inequalities can be verified:

$$\frac{(2q-1)m}{2n} \leq p < \frac{(2q+1)m}{2n}$$

§3. Stair Code. The object shown in figure 2 is a segment drawn by any one of the algorithms mentioned in the introduction and in particular by Bresenham's algorithm.

As can be seen, it looks like a staircase whose steps are generally different in size. With this type of structure, the last pixel of each step determines the rest. With this idea in mind, the list, made up of both segment end points and the last points of each step, can be interpreted as a code. Notice that from the previous proposition we can deduce that the last pixel on any step is (h_i,i) where $h_i = \lfloor (2i+1)m/2n \rfloor$, from $i = 0$ to $n-1$. Nevertheless, observe that the ordinate of the end of a step is one greater than the ordinate of the end of the previous one, so we need only calculate the abscissae of these points to obtain a code. Therefore, the *stair code* of the segment $s((p_1,q_1),(p_2,q_2))$ can be defined as the list $((p_1,q_1),(p_2,q_2), h_0, h_1,\ldots, h_{n-1})$, where $0 < n = q_2-q_1 < m = p_2-p_1$ and $h_i = \lfloor (2i+1)m/2n \rfloor$. It is easy to see that this code uses only O(n) space; that needed to store the n+4 integers which make it up.

§4. A new drawing algorithm. Bresenham's algorithm is an incremental algorithm which calculates a pixel from the information given by the one before. The new algorithm that we present also draws the pixels in an incremental way, but only needs to perform arithmetical calculations to find the ends of the steps. To be exact, this algorithm calculates the stair code of the segment which is drawn although it does not store it.

In order to reach this objective we consider the following equalities: $dn + r = m$ with $0 \leq r < n$, $h_i 2n + r_i = (2i+1)m$ with $0 \leq r_i < 2n$, where d and r (respectively, h_i and r_i) are the quotient and the rest of the integer division of m by n (respectively, of $(2i+1)m$ by $2n$). From these equalities we can deduce that if $r_i < 2n-2r$ then $h_{i+1} = h_i + d$ and $r_{i+1} = 2r + r_i$, otherwise $h_{i+1} = h_i + d + 1$ and $r_{i+1} = 2r + r_i - 2n$.

The algorithm we propose, from $i = 1$ to $n-1$, calculates the value h_i and updates the variable of decision r_i from the value h_{i-1} and the variable of decision r_{i-1}. To calculate h_0 and r_0 we can simply observe that $h_0 2n + r_0 = dn + r$. Therefore, if d is even $h_0 = d/2$ and $r_0 = r$, and if d is odd, $h_0 = (d-1)/2$ and $r_0 = r + n$.

In the figure 4 we show an implementation, in PASCAL, of the new algorithm.

It can be seen that this algorithm does not exactly illuminate the pixels that Bresenham's algorithm does. They coincide in those points (p,q) which verify the inequality $| q - np/m | < 1/2$. However, if the points (p,q) and (p,q-1) are equidistant from the segment, Bresenham's algorithm chooses the former, while NEW takes the latter. As we have already mentioned, this difference is not important as the choice of one pixel or another is by convention. In any case, a slight modification of NEW would draw the same pixels as Bresenham's algorithm would.

To compare the efficiency of both algorithms we can calculate their costs depending on the number of additions and subtractions they perform. This is a realistic measure since both draw the same number of pixels and use integer arithmetic. It is not difficult to check that the cost of Bresenham's algorithm is $2m+n$ while the NEW cost is $m+3n-2$. Therefore, both algorithms have the same worst-case asymptotic complexity. However,

from these calculations we can deduce that Bresenham's algorithm is faster for the segments whose slope is greater than 1/2, while the one we have just presented is more appropriate for those whose slopes are less than 1/2.

```
procedure NEW(n,m:integer);
var incr1_hi,incr1_ri,incr2_hi,incr2_ri,rcom,hi,ri,x,y:integer;
begin
    incr1_hi:=m div n; incr1_ri:=m mod n;      {incr1_hi*n + incr1_ri = m}
    hi:=incr1_hi div 2;                        {calculation of h0}
    if hi*2=incr1_hi                           {calculation of r0}
      then ri:=incr1_ri
      else ri:=incr1_ri+n;
    incr1_ri:=2*incr1_ri;                {increase for the rest in the case ri < 2n - 2r}
    incr2_hi:=incr1_hi+1;                {increase for hi in the case ri ≥ 2n - 2r}
    incr2_ri:=incr1_ri-2*n;              {increase for the rest in the case ri ≥ 2n - 2r}
    rcom:=-incr2_ri;
    x:=0; y:=0;
(1) while x≤hi do                        {drawing the first step}
      begin write_pixel(x,y); x:=x+1 end;
    y:=y+1;
(2) while y<n do
      begin
        if ri<rcom
          then begin hi:=hi+incr1_hi; ri:=ri+incr1_ri end      {ri < 2n - 2r}
          else begin hi:=hi+incr2_hi; ri:=ri+incr2_ri end;     {ri ≥ 2n - 2r}
(3)     while x≤hi do                  {drawing the intermediate steps}
          begin write_pixel(x,y); x:=x+1 end;
        y:=y+1
      end;
(4) while x≤m do                         {drawing the last step}
      begin write_pixel(x,y); x:=x+1 end
end;  {end of NEW}
```

Figure 4. An algorithm for drawing a segment

In practice, the difference between the performance of both these algorithms is hardly noticeable. As proof of this, we have implemented both algorithms on an HP-APOLLO 9000/425t computer, calculating the pixels corresponding to segments s((0,0),(m,n)), for some values of m between 2 and 4000, and $1 \leq n \leq m$. After running each algorithm a million times the difference in speed observed was a few milliseconds in favour of one or another.

So, in practice it is not important whether we choose one algorithm or another to draw a segment. However, in certain situations, the NEW algorithm seems more convenient because it is based on a code which contains more explicit information on the segment points. On the other hand, the NEW algorithm can be modified so that, without any additional expense, pixels of the segments whose slope is more than 1/2 can be drawn by means of calculation of the pixels of the symmetric segment with regard to the main diagonal. In this way we obtain an algorithm whose running cost is less than Bresenham's whatever the slope of the segment may be.

Notice that the NEW algorithm may be modified to obtain the stair code of a segment from its end points. The only thing we need to do is replace the pixels of the segment in the output with this code. In this situation this calculation can be carried out in 3n steps, while using Bresenham's algorithm it would take 2m+n steps.

The algorithm shown draws a segment setting out from its end points. Nevertheless, should the stair code of this segment be already calculated, we can obtain an algorithm for drawing from the previous one, by eliminating the calculation of the code from it. In this case the cost of this algorithm is only m+n.

§5. The BELONG problem. We shall give the algorithm that solves the BELONG problem as an example of the use of the stair code. To decide whether a point belongs to a segment we need only check if it is on the step possibly determined by its ordinate. In order to do this it suffices to check if the ordinate is in the permitted range and then to see if the abscissa is on the corresponding step; that is, if $h_{y-1} < x \leq h_y$ is fulfilled. With this idea in mind, figure 5 shows a function written in pseudocode which uses the stair code of the segment and solves the problem.

```
Function BELONG(x,y:integer; cod:code):boolean;
begin
 if (y<y1) or (y>y2)
        then return(false)
        else if y=y1
                    then return((x1≤x) and (x≤x1+h0))
                    else if y=y2
                                then return((x1+hn-1<x) and (x≤x2))
                                else return((hy-1<x-x1) and (x-x1<hy))
end;
```

Figure 5. An algorithm that decides whether a point belongs to a segment from its stair code

The time complexity of this algorithm is constant and it uses only sum and comparison operations because the values of h_{y-1} and h_y appear explicitly in the code. Next we state the solution to this problem by the following proposition whose proof is easily deduced from the algorithm and the previous comments.

Proposition 2. The BELONG decision problem can be answered in constant time and without preprocessing, using integer products and divisions. This problem can be solved in constant time, given O(n) space and O(n) preprocessing time, using only comparisons.

§6. Digitization. Rosenfeld-Kim [11] presented an accurate notion of digital segment if the 8-adjacency is considered on the grid $\mathbb{Z}^2$. To be more precise, they define a digital segment as a digital 8-arc which is the digitization, by means of what they call 'grid digitization', of an Euclidean segment. After this, they give an intrinsic characterization of

the digital 8-arcs which are digital segments. This notion, as we shall soon see, can be given without using the previous concept of a digital arc.

An *Euclidean digitization* is a function c: $\mathbb{R}^2 \longrightarrow \mathcal{P}(\mathbb{Z}^2)$ such that c(p,q) = (p,q) for every (p,q) $\in \mathbb{Z}^2$ and $\{\tilde{c}(p,q);\ (p,q) \in \mathbb{Z}^2\}$ is a locally finite family of connected subspaces with compact closure of the Euclidean plane $\mathbb{R}^2$, where $\tilde{c}(p,q) = \{(x,y) \in \mathbb{R}^2;$ c(x,y) = (p,q)} and $\mathcal{P}(\mathbb{Z}^2)$ stands for the set of all subsets of $\mathbb{Z}^2$ (see [5] for a more general definition). A *digitization of a subspace* of $\mathbb{R}^2$ is its image through this function c. A digitization of an Euclidean straight line (segment) is called a *digitized straight line* (*segment*, respectively).

If one chooses the $\tilde{c}(p,q) = \{(x,y);\ |\ p - x\ | + |\ q - x\ | \leq 1/2\}$ digitization, it is easy to prove that the digitization of an Euclidean straight line $r_{(m,n)}$ which passes through the origin (0,0) and the point (m,n), where $0 < n < m$, is the digitized straight line

$$L_{(m,n)} = \mathbb{Z}^2 - \{(p,q) \in \mathbb{Z}^2;\ \tilde{c}(p,q) \subset \mathbb{R}^2 - r_{(m,n)}\}$$

When the Euclidean straight line $r_{(m,n)}$ goes through a point like (p,q+ 1/2), where (p,q) $\in \mathbb{Z}^2$, the digitized straight line $L_{(m,n)}$ contains the points (p,q) and (p,q+1), so that, in this sense, the corresponding digital object looks thick in these points. A *digital straight line* (or a *digital segment*) is the object obtained on straightening these zones. To be more precise, in the case before the associated digital straight line is

$$R_{(m,n)} = L_{(m,n)} - \{(p,q) \in L_{(m,n)};\ (p,q-1) \in L_{(m,n)}\}$$

In other words, the straightening of the digitization of an Euclidean straight line (segment) is called *digital straight line* (*digital segment*, respectively). With this definition it can be shown that the NEW algorithm presented in this paper is a drawing algorithm for digital segments and that the stair code of a segment encodes the associated digital segment.

Acknowledgements. We would like to acknowledge the partial assistance of the Diputación General de Aragón (D.G.A) (for the work of Angel Francés) and the Dirección General de Investigación Científica y Técnica (DGICYT) and the Junta de Andalucía (for the work of Alberto Márquez).

References

[1] J.E. Bresenham, Algorithm for Computer Control of a Digital Plotter, *IBM Systems Journal* 4 (1), 25-30, 1965.

[2] R. Brons, Linguistic Methods for the Description of a Straight Line Upon a Grid, *Computer Graphics and Image Processing* 3 (1984), pp. 48-62.

[3] C.M. Castle, M.L.V. Pitteway, An Application of Euclid's Algorithm to Drawing Straight Lines, in *Proc. NATO ASI on Fundamental Algorithms for Computer Graphics*, Springer-Verlag, Heidelberg, 1985.

[4] C.M. Castle, M.L.V. Pitteway, A Technique for Encoding 'Best-Fit' Straight Lines, *Computer Journal*, vol. 30, no. 2, pp. 168-175, 1987.

[5] E. Domínguez, A. Francés, A. Márquez, Digital Spaces: A Mathematical Model for Digital Image Processing, *to appear*..

[6] L. Dorst, R. Duin, A Framework for Calculations on Digitised Straight Lines, *IEEE Trans. Pattern Analysis and Machine Intelligence* PAMI-6 (5), 1984.

[7] R.A. Earnshaw, Line Tracking with Incremental Plotters, *Computer Journal* 23 (1), 1980.

[8] J.D. Foley, A. Van Dam, *Fundamentals of Interactive Computer Graphics*, The Systems Programming Series, Addison-Wesley, 1984.

[9] H. Freeman, Boundary Encoding and Processing, in *Picture Processing and Psichopictorics*, 241-266, Academic Press, New York, 1970.

[10] M.L.V. Pitteway, A. Green, Bresenham's Algorithm with Run-Line Coding Shortcut, *Computer Journal* 25 (1), 114-115, 1982.

[11] A. Rosenfeld, Ch.E. Kim, How a digital computer can tell whether a line is straight, *Amer. Math. Monthly* 89, 230-238, 1982

[12] A. Rosenfeld, R.A. Melter, Digital Geometry, *The Mathematical Intelligencer* 11 (3) 69-72, 1989

Druida: Una Herramienta Basada en Conocimiento para el Afinamiento de Sistemas Unix

Mario Droguett C. , Magister U.C.
Ignacio Casas R., Ph.D.
Francisco Aurtenechea O., M.Phil.
Depto. de Ciencia de la Computación
Escuela de Ingeniería
Pont. Universidad Católica de Chile
Casilla 6177, Santiago Chile

RESUMEN

Este documento describe el análisis, diseño e implementación de un prototipo de sistema experto para el afinamiento del sistema operativo Unix. El paradigma de afinamiento se basa tanto en reglas de cotas de rendimiento, como en modelos analíticos de redes de espera para la evaluación y predicción del rendimiento. El objetivo principal de este trabajo ha sido el estudiar la automatización del afinamiento de un sistema computacional, considerando la identificación de los problemas de rendimiento y de los parámetros a modificar. Se describe el espacio de búsqueda del sistema experto y la combinación de estrategias de búsqueda y modelación.

Palabras Claves: Evaluación del Rendimiento, Afinamiento, Sistemas Expertos.

1. Introducción

La tecnología de los sistemas computacionales modernos demanda recursos escasos y costosos, requiriéndo de procesos continuos de afinamiento, periódicas evaluaciones y rigurosas planificaciones de capacidad, para asegurar un uso eficiente y óptimo que satisfaga los crecientes requerimientos de los usuarios.

Con la creciente complejidad de los sistemas computacionales de aplicación y la proliferación de tales sistemas, el problema de la evaluación del rendimiento y el afinamiento de éstos es una tarea más difícil y llega a ser realizada por profesionales que no se encuentran, en general, lo suficientemente preparados para llevarlas a cabo.

Existe un gran número de excelentes herramientas disponibles para recolectar y manejar información sobre el rendimiento de un sistema. Mediante el examen de la información provista por éstas y comparándola con ciertos valores esperados, es factible identificar un problema de rendimiento y proponer una solución. Sin embargo, esto requiere de una gran cantidad de experiencia y conocimiento para interpretar la, generalmente, muy voluminosa información entregada por las herramientas.

Esta situación satisface, por lo tanto, los requerimientos básicos para la especificación del dominio apropiado para un Sistema Experto[1] [5] [8] [11].

Un Sistema Experto para afinamiento de sistemas computacionales debe proveer las siguientes funciones básicas [6]:

a. Recolectar y Resumir datos de monitoreo del sistema.
b. Correlacionar datos relacionados y no relacionados.
c. Aplicar reglas y juicios de valor sobre los datos.
d. Emitir conclusiones sobre los datos.
e. Recomendar cambios para mejorar el rendimiento.

Las primeras dos funciones han sido cubiertas por numerosas herramientas de monitoreo desarrolladas durante los últimos veinte años [10]. Además, actualmente existen tanto aplicaciones tradicionales, como Sistemas Expertos que abarcan las tres últimas funciones.

---

[1] Los términos *Sistema Experto* y *Sistema basado en Conocimiento* son utilizados como sinónimos en este documento.

Sin embargo, no existe una herramienta computacional que abarque las cinco funciones para diferentes ambientes computacionales [9]. En la práctica es necesario un experto.

En este trabajo se estudia la utilización de Modelos de Redes de Espera en un Sistema Experto de Afinamiento, identificando las ventajas y desventajas de su utilización.

Deseamos determinar la factibilidad de utilizar Modelos Analíticos de Redes de Espera en un Sistema Experto de Afinamiento, obteniendo así una cuantificación del efecto de su utilización para la Evaluación del Rendimiento de sistemas computacionales Unix.

Para comprobar nuestro objetivo hemos desarrollado un prototipo de Sistema Experto, llamado Druida, el cual nos ha permitido analizar el efecto de la utilización de Modelos de Redes de Espera en un Sistema Experto de Afinamiento.

A continuación se describen los conceptos básicos de afinamiento y transmisión del conocimiento. La siguiente sección describe a Druida, los objetivos de su desarrollo, su diseño y cómo caracteriza al sistema. Las secciones siguientes describen el conocimiento manejado por Druida y cómo éste es representado. Se entregan los resultados obtenidos al aplicar el prototipo a un problema real y, finalmente, las conclusiones son acompañadas con una visión del trabajo futuro en el área.

2. Concepto de Afinamiento.

Por afinamiento de sistemas computacionales entendemos, dada la carga de trabajo que el sistema enfrenta, el ajustar ciertos parámetros para obtener un mejor rendimiento.

El trabajo de afinamiento es generalmente realizado a través del uso de un método heurístico, que, en realidad, es una síntesis de reglas formalizadas y experiencia.

Esto no garantiza el encontrar una buena solución al problema, ya que, generalmente, la experiencia es escasa y las reglas no son suficientes. Por último, dada la decisión de solucionar un problema, no es sencillo predecir el efecto real de ésta.

Así, para la construcción de nuestro Sistema Experto, que llamaremos Druida, las reglas son complementadas con modelos cuantitativos. El efecto de esta estrategia, es la capacidad de Druida de predecir el efecto de las recomendaciones que realiza el sistema.

3. Druida.

A continuación se presentan los principios de diseño y consideraciones de evaluación del rendimiento contempladas en la construcción del Sistema Experto Druida.

Druida es una herramienta orientada a apoyar el afinamiento de sistemas Unix System V. Druida, contiene conocimiento sobre Unix, análisis de datos y modelación de sistemas computacionales para:

a. Diagnosticar un problema de rendimiento.
b. Ubicar áreas que puedan ser modificadas para mejorar el rendimiento.
c. Hacer recomendaciones para el afinamiento del sistema.

3.1 Objetivos en el desarrollo de Druida.

La finalidad principal de Druida es actuar como una herramienta para el afinamiento de sistemas computacionales Unix System V.

Para realizar lo anterior obtiene de forma automática las medidas de rendimiento actuales del sistema y, utilizando el conocimiento experto almacenado en su base de conocimientos, determina las modificaciones necesarias para mejorar su rendimiento.

Para llevar el rendimiento de un sistema computacional a un nivel satisfactorio, dos estrategias principales pueden llevarse a cabo: afinar el sistema actual y/o comprar hardware adicional. [16] [2] [3]

El afinamiento es normalmente la opción más barata y debe ser realizada primero. Si un sistema no ha sido recientemente afinado, un considerable aumento en su rendimiento puede ser obtenido a un bajo costo. [3]

3.2 Diseño Conceptual

Para el desarrollo de Druida se elige la orientación a Objetos como paradigma de diseño y programación. Druida se ve como una colección de objetos que realizan tareas específicas y se comunican para obtener información del sistema, requerir información del usuario, realizar el

diagnóstico, y entregar sugerencias para mejorar el rendimiento del sistema. En la figura 1 se muestran las relaciones entre los distintos objetos que conforman Druida.

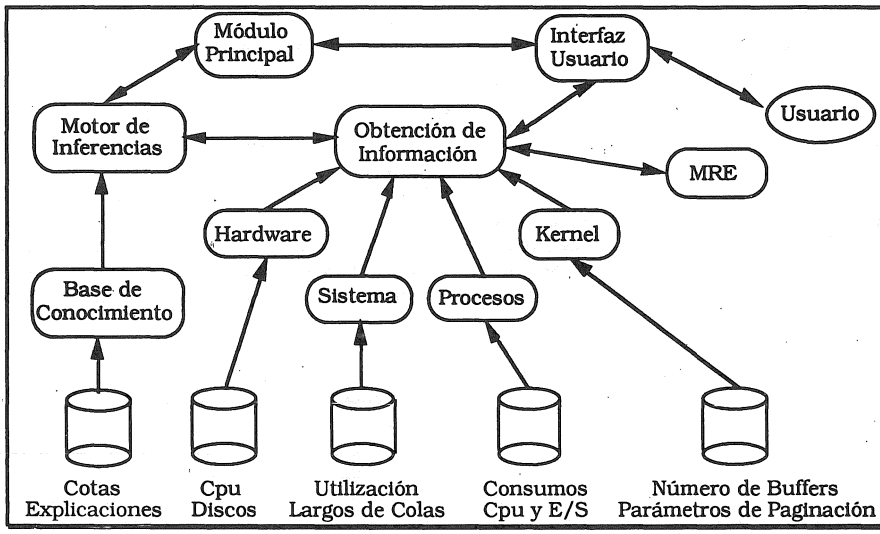


Figura 1. Relaciones entre los Objetos de Druida.

Dada la selección de este paradigma, se utiliza para la implementación de Druida el lenguaje de programación C++[2]. El uso de C++ radica principalmente en la posibilidad de reutilizar otras herramientas ya desarrolladas, la eficiencia del código generado y en la productividad de las tareas de diseño y desarrollo de prototipos.

Los distintos objetos que conforman Druida proveen entonces una serie de métodos para comunicarse con otros objetos y para realizar sus tareas.

4. Adquisición del Conocimiento

El conocimiento necesario para realizar el afinamiento de un sistema computacional es obtenido de múltiples fuentes, siendo las principales los manuales del sistema, publicaciones sobre el tema, opiniones de expertos y experimentación.

En la interacción con expertos se trabajó principalmente con expertos de UNISYS CHILE y administradores experimentados del Departamento de Ciencia de la Computación. La interacción con los primeros se basó principalmente en la discusión de los resultados obtenidos por el prototipo inicial (basado en la información de manuales), y con los segundos se trabajó en la prueba de la Base de Conocimientos y la ampliación de las fuentes de información. Para mayores detalles en este caso específico refiérase a [12].

Los manuales del sistema entregan información que es específica al sistema en estudio. En este sentido proveen procedimientos para detectar y solucionar problemas, así como recomendaciones para aumentar el rendimiento del sistema. Estas recomendaciones son generales, por lo cual deben ser adecuadas al sistema en estudio para ser útiles.

Las publicaciones sobre el tema y las opiniones de expertos no siempre se refieren a sistemas Unix[3]. Es por ésto que la información obtenida de ellos debe ser verificada y ajustada para ser utilizada en un sistema Unix. Para validar esta información se utiliza experimentación.

Los experimentos consisten en medir el sistema, aplicar las medidas obtenidas a un criterio propuesto[4], por ejemplo, para un sistema VMS, y obtener una recomendación sobre la

---

[2] Para información sobre C++ refiérase a [22].

[3] Para publicaciones sobre Unix System V, refiérase a [21].

[4] En [24] una serie de criterios son presentados para diversos tipos de sistemas.

base de este criterio. Luego, se procede a efectuar la modificación recomendada en el sistema Unix, y éste es medido nuevamente. Los resultados obtenidos son analizados para determinar la validez del criterio propuesto, de modo de ajustarlo a un sistema Unix, o determinar que este criterio no es aplicable.

4.1. Parámetros de Afinamiento.

Sobre la base de los objetivos que dan partida al desarrollo de Druida, inicialmente es necesario identificar aquellos parámetros relativos al desempeño de un sistema Unix que son medibles y modificables de manera automática.

La primera tarea a realizar para efectuar un afinamiento es detectar y diagnosticar el problema de rendimiento que éste presenta. Identificado el dispositivo que produce la baja en el rendimiento, una reducción en su carga de trabajo puede aumentar sensiblemente el rendimiento. [6]

4.2. Medidas de Rendimiento.

Una vez identificados, tanto los parámetros modificables automáticamente como los que deben ser modificados de forma manual, y conocida la forma de reconstruir automáticamente un sistema Unix, se deben obtener medidas de rendimiento del sistema en estudio, los valores actuales de los parámetros del kernel y la configuración de hardware del sistema.

Del cúmulo de información de rendimiento que se puede obtener mediante los utilitarios descritos en [23], es necesario definir aquella que es relevante para el estudio del sistema que realiza Druida.

Por experiencias previas en estudios de modelación y opiniones vertidas en publicaciones del área, se eligen como medidas relevantes del rendimiento del sistema Unix las siguientes:

Medida	Componente de Aplicación
Utilización	Cpu, Discos.
Largos de Cola	Cpu, Discos.
Demanda de servicio	Clases de procesos sobre Cpu y Discos.
Actividad de Paging & Swap	Memoria y Partición de swap en disco.
Tiempo de Respuesta	Clases de procesos sobre el Sistema completo.

Además de las medidas señaladas anteriormente, es necesaria información adicional sobre el comportamiento del sistema para realizar el diagnóstico y determinar una solución al problema. Esta información corresponde, por ejemplo, al número de páginas libres en memoria, necesaria para determinar si un aumento en el número de "buffers" del sistema producirá como efecto lateral un aumento en la actividad de swap.

4.3. Utilización de Modelos de Redes de Espera.

En la evaluación del rendimiento de sistemas computacionales, los Modelos de Redes de Espera son la principal herramienta utilizada para analizar y predecir el rendimiento de un sistema computacional [3] [8] [19].

Un Modelo de Redes de Espera (MRE) es una representación abstracta de un sistema computacional, que se apoya en la definición de las Clases de Procesos (Clientes), unidades (Centros de Servicio), y Parámetros de Carga de Trabajo (Tasas de Llegada, Tiempo de Servicio, etc) [3].

El modelo es evaluado por medio de técnicas analíticas (resolución de sistemas de ecuaciones basados en teoría de colas), para obtener una evaluación del desempeño o rendimiento del sistema (Utilización de la CPU, Tiempo de Respuesta, Caudal y Largos de Cola) [16].

Los MRE permiten evaluar el rendimiento del sistema, identificando las causas que limitan este rendimiento. Utilizando MRE podemos, entonces, realizar las tareas de detección y diagnóstico que son deseables en un Sistema Experto.

El uso de MRE permite, además, determinar las formas de mejorar este rendimiento, focalizando el esfuerzo de afinamiento [4]. Apoyamos, así, la determinación de recomendaciones para solucionar los problemas detectados, al focalizar las áreas de análisis.

En los MRE las modificaciones sugeridas para el afinamiento pueden ser reflejadas en el modelo del sistema. Evaluando el modelo se puede predecir el rendimiento futuro del sistema, luego de las modificaciones.[3].

Druida incorpora MRE en su diseño de manera de acceder a las ventajas señaladas anteriormente. Así, la información obtenida del sistema es manejada por Druida desde un nivel de abstracción mayor que el que proveen los utilitarios del sistema.

5. Representación del Conocimiento

Para la representación del conocimiento se elige un esquema basado en reglas. El conjunto de las reglas definidas es utilizado por el Motor de Inferencias para realizar el afinamiento del sistema. A continuación se describen el Motor de Inferencias de Druida, mostrando ejemplos de la representación del conocimiento utilizada, y su relación con los mecanismos de explicación provistos.

5.1 El Motor de Inferencias.

Desde el punto de vista de los objetos que conforman Druida, el Motor de Inferencias de éste no es más que otro objeto con el cual pueden comunicarse, recibir y entregar datos. Mas, en este caso, se incorpora, como Motor de Inferencias de Druida, el motor de inferencias de una herramienta disponible en el mercado, la herramienta incorporada corresponde a KES[5]. La selección de KES se basa principalmente en claridad de uso y disponibilidad para experimentación.

Para incorporar el motor de inferencias en un programa escrito en C++, KES provee una librería de funciones que permiten manejar la información que entrega y/o recibe el motor de inferencia, y controlar el proceso de razonamiento. Provee mecanismos de explicación simples que pueden ser manejados y ampliados por el programa que controla el Motor de Inferencias.

5.2 La Base de Conocimientos.

La Base de Conocimientos de Druida se encuentra escrita en el lenguaje provisto por KES para su Motor de Inferencias. Se representa en ella sobre la base de reglas y comprende dos áreas principales de conocimiento relativas al afinamiento de sistemas Unix. Estas áreas corresponden a Diagnóstico y Sugerencias. Las áreas de Diagnóstico y Sugerencias proveen reglas para:

- **a.** Realizar la identificación de el o los posibles problemas existentes en el sistema en estudio.
- **b.** Obtener los datos necesarios desde el sistema o el usuario y entregar el diagnóstico realizado.
- **c.** Sugerir, dado el problema determinado, la solución más adecuada al problema detectado.

5.3 Reglas y Modelos de Redes de Espera.

Para analizar como se relacionan los Modelos de Redes de Espera y la Base de Conocimientos descrita en base a reglas utilizaremos el siguiente ejemplo:

Supongamos que el motor de inferencias ha determinado que el cuello de botella del sistema en estudio corresponde al sub-sistema de disco. Luego el motor de inferencias intentará determinar una solución a este problema. Para esto, por ejemplo, tratará de satisfacer la siguiente regla:

```
if Balance Disk Tuning = yes          (1)
then
Disk Tuning = Balance Disk.
endif.
```

Esta regla puede interpretarse de la siguiente manera: Para ver si el valor de esta regla es verdadero Druida intentará obtener el valor de la variable **Balance Disk Tuning**, para ésto comparará el valor obtenido con la constante **yes** y, si esto es efectivo, concluirá que se puede obtener una mejora en el rendimiento del sistema realizando un balance de carga en el sub-sistema de disco.

---

[5] KES Knowledge Engineering System. Copyright 1990 Software Architecture & Engineering, Inc. Distribuido por UNISYS Corp.

Para obtener el valor de la variable **Balance Disk Tuning** el motor de inferencias utilizará entonces la siguiente regla:

```
if MRE Balance Disk Test le 0.9          (2)
then
Balance Disk Tuning = yes.
      endif.
```

Para obtener a su vez el valor de **MRE Balance Disk Test** Druida utilizará el siguiente cálculo:

```
MRE Balance Disk Test: real
  (3)
[default: (MRE Balance Disk Response Time) / (MRE Response Time)].
```

Luego, para realizar el cálculo el motor de inferencias debe obtener los valores de las variables involucradas, que en este caso corresponden a **MRE Balance Disk Response Time** y **MRE Response Time**. Para obtener estos valores el motor de inferencias realiza las siguientes consultas:

```
MRE Response Time: real
    (4)
{question: "Ingrese MRE Response Time:",
   "MRE","RTIME"}.
```

```
MRE Balance Disk Response Time: real          (5)
{question:
   "Ingrese MRE Balance Disk Response Time:",
   "MRE","BDISK"}.
```

Estas consultas son generadas por el objeto Motor de Inferencias y enviadas como mensajes al objeto Módulo de Obtención de Información. Este identifica las consultas y las envía al objeto MRE, como se muestra en la figura 2.

Para la Primera consulta (4) el objeto MRE construye un Modelo de Redes de Espera del sistema en estudio[6], enviando para ésto mensajes al módulo de obtención de información para determinar las componentes del modelo y los valores de los parámetros que lo definen. El módulo de obtención de información genera a su vez mensajes a los objetos Hardware, Sistema, Procesos y Kernel para obtener los datos solicitados (ver figura 1).

Luego el modelo construido es resuelto y validado por el objeto MRE[7]. Luego de validar el modelo el valor del tiempo de respuesta promedio persivido por las clases del modelo es retornado al motor de inferencias como valor de la variable **MRE Response Time**.

Para resolver la siguiente consulta (5) el objeto MRE utiliza el modelo original del sistema definido en la consulta anterior, pero en éste, la carga de trabajo al sub-sistema de discos es distribuida de manera uniforme entre los discos que conforman el sub-sistema.

Luego el modelo es resuelto, y el valor del tiempo de respuesta promedio persivido por las clases del modelo modificado (balanceado en disco) es retornado al motor de inferencias como valor de la variable **MRE Balance Disk Response Time**.

Una vez obtenidos estos valores el Motor de inferencias realiza el cálculo indicado en (3), el cual define el valor de la variable **MRE Balance Disk Test**, si este valor es menor que 0.9 indica que la diferencia entre el tiempo de respuesta persibido por un usuario promedio del sistema, y el que persiviría luego de realizado una balance de carga en el sub-sistema de disco tiene una reducción mayor al 10%.

---

[6] Una vez construido el modelo éste se mantiene dentro de Druida. Posteriormente este modelo puede ser modificado y re-evaluado.

[7] Se utilizan para ésto los algoritmos de Análisis de Valor Medio descritos en [16].

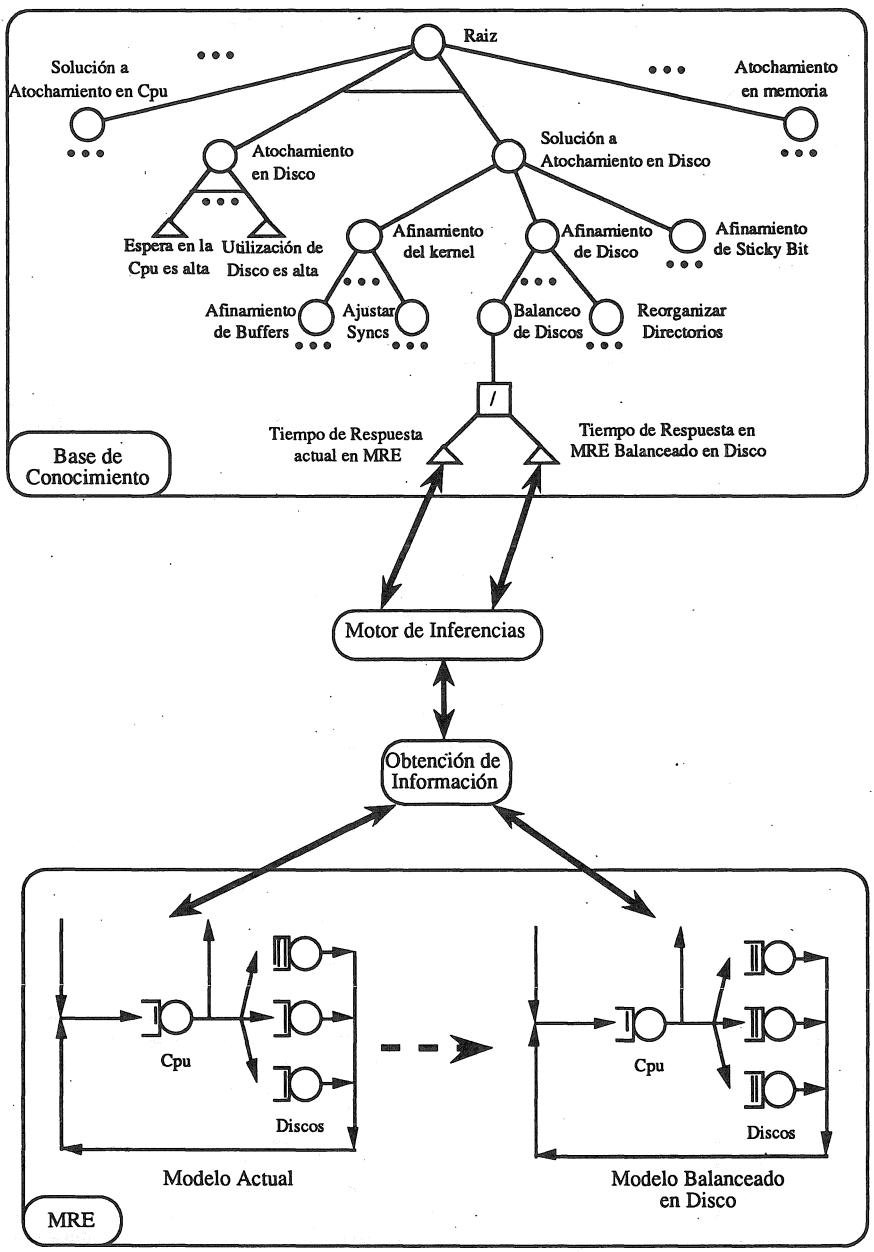


Figura 2. Realación entre la Base de Conocimientos y los Modelos de Redes de Espera.

La condición anterior es verificada por la regla (2), lo que permite hacer verdadera la regla (1), la cual hace al motor de inferencias concluir que una mejora al rendimiento puede ser obtenida mediante un balance de carga en el sub-sistema de disco.

6. Caracterización del Sistema y su Carga de Trabajo.

En Unix System V existe una serie de utilitarios que recolectan y generan informes sobre las medidas de rendimiento del sistema en estudio [20] [17]. Ejemplos de estos utilitarios son el Generador de Reportes de la Actividad del Sistema sar(1)[8], y acctcom(1), que entrega información sobre los procesos ejecutados en el sistema.

La información provista por estas rutinas puede ser utilizada para caracterizar la carga de trabajo del sistema y analizar su rendimiento, pero ésta es muy voluminosa y requiere invertir gran cantidad de trabajo para que llegue a ser útil.

Por esto Druida se apoya, para procesar esta información, en las facilidades que provee una herramienta desarrollada por el Departamento de Ciencia de la Computación de la Universidad Católica: Asterix[9].

Asterix es utilizado para realizar la caracterización de la carga de trabajo del sistema y obtener las medidas de rendimiento del mismo. Druida utiliza la información que entrega Asterix tanto para diagnosticar el sistema en estudio, como para entregar recomendaciones para mejorar su rendimiento.

Cuando el Motor de Inferencia de Druida requiere, por lo tanto, algún dato específico en su cadena de razonamiento, y éste no puede ser inferido a partir de otros, genera un mensaje al módulo de obtención de información. Este, a su vez, obtiene el dato requerido desde el sistema.

7. Resultados obtenidos

Se describe a continuación la utilización de Druida en el afinamiento de un sistema Unix. El sistema es descrito en cuanto a sus componentes de hardware y la carga inicial que enfrenta.

El sistema en estudio es 'Puyehue', un equipo UNISYS U6000/55 instalado en el Departamento de Ciencia de la Computación de la Universidad Católica de Chile. Puyehue cuenta con un sistema operativo Unix System V versión 3.2, y se encuentra dedicado a investigación. Las características técnicas de Puyehue son las siguientes:

- **a.** Cpu INTEL 80386 a 33 Mhz.
- **b.** Procesador matemático esclavo Weitek.
- **c.** 16 Mb de Memoria Principal.
- **d.** 1 unidad de Cinta.
- **e.** 3 discos SCSI que totalizan 1,2 Gb.
- **f.** 8 terminales uvt1224.
- **g.** 10 80386sx conectados en red Ethernet.
- **h.** Conexión Ethernet a la Red Interdepartamental de la Escuela de Ingeniería.

Los días 8, 10 y 15 de Octubre de 1991 fueron presentados a Druida, para la obtención de recomendaciones para mejorar el rendimiento. Cada día es monitoreado desde las 9:00 horas hasta las 19:00 horas.

Al analizar la información de rendimiento de estos días, y resolver los modelos asociados, Druida señala que no existen atochamientos importantes. Luego, indica que el rendimiento puede ser mejorado efectuando un balanceo de la carga que enfrentan los discos.

Druida no recomienda en este caso modificaciones a los parámetros del kernel, ya que su modificación no tiene un impacto significativo en el rendimiento.

Para cuantificar objetivamente el efecto del afinamiento sugerido por Druida se utiliza un experimento[10] que permite determinar el tiempo de respuesta percibido por el usuario para un comando estandar del sistema.

---

[8] Para información sobre los nombres de comandos en Unix System V refiérase a [23]

[9] Asterix es una herramienta de apoyo al Análisis, Supervisión, y Evaluación de Rendimiento de Sistemas Unix. Permite monitorear y obtener información del sistema en estudio, enfocada ésta hacia un estudio de Modelación. Para mayor información refiérase a [4].

[10] Este experimento se encuentra propuesto en [18] y es utilizado para analizar el cambio en el rendimiento de un sistema VAX11-780 con sistema operativo Unix.

En este caso el comando estandar corresponde a la compilación en C++ de un trozo de código de aproximadamente 700 líneas. El experimento consiste en realizar una compilación de este módulo cada 5 minutos, utilizando para ello el utilitario crontab de Unix. Mientras se realiza la prueba el sistema mantiene u operación normal, atendiendo otros procesos y usuarios.

Para medir el tiempo de respuesta se utiliza el comando timex de Unix, el cual permite obtener tanto el tiempo de respuesta que percibe el usuario, como los consumos del proceso en el sistema.

Luego de realizado el afinamiento, el sistema es medido nuevamente los días 29 y 30 de Octubre, y el día 20 de Noviembre. Cada día es monitoredo desde las 9:00 horas hasta las 19:00 horas.

Analizaremos los resultados obtenidos del afinamiento mediante la técnica de gráficos de "Caja y Bigote" o "Centro y Periferia" ("Boxplot" o "Box and whiskers plot") [13] [15].

Esta técnica permite representar gráficamente los datos obtenidos de un modo coherente, cuidando que los valores extremos obtenidos en los datos no afecten la percepción del fenómeno en estudio.

En la figura 3 se compara, mediante gráficos de caja y bigote, los tiempos de respuesta del sistema a las compilaciones antes y después del afinamiento propuesto por Druida. Para cada día de observación se presentan los valores mínimo, medio y máximo de respuesta del sistema a las compilaciones, medidos en segundos. Se representa además en el gráfico las utilizaciones de Cpu promedio de cada día.

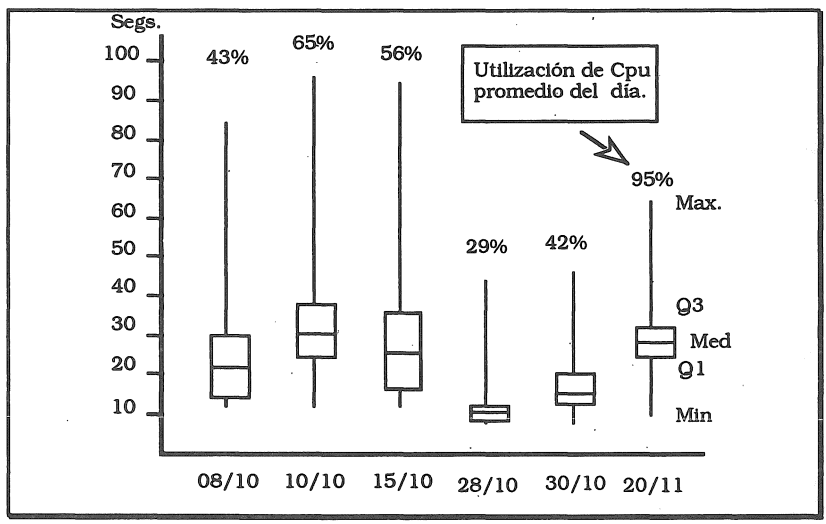


Figura 3. Respuesta del Sistema antes y después del Afinamiento.

Vemos así que el afinamiento propuesto por Druida ha producido una reducción visible del tiempo de respuesta del sistema. Si se comparan los días 8 y 30 de Octubre podemos ver que para utilizaciones promedio de Cpu similares el tiempo de respuesta del sistema es menor, y la dispersión del mismo es también menor.

Al analizar el día 20 de Noviembre vemos que al enfrentar el sistema una utilización de Cpu muy superior la respuesta del sistema es en promedio similar a la obtenida para utilizaciones entre 50 y 60%, pero la dispersión de estas respuestas es muy inferior.

Esto nos permite señalar entonces que el efecto del afinamiento realizado ha llevado a una reducción notable en el tiempo de respuesta del equipo a las compilaciones. El análisis anterior es descrito en mayor detalle en [12].

8. Conclusiones

Se ha descrito Druida, una herramienta basada en conocimiento para el afinamiento de sistemas computacionales Unix System V.

Se ha observado, durante el uso de la herramienta como prototipo, que ésta no sólo ayuda a operadores y administradores con poca experiencia, sino que también ayuda a operadores de sistema expertos.

Al obtener información útil desde grandes volumenes de datos, y entregar medidas de rendimiento comprensibles sobre los sistemas y al liberarlos de experimentaciones innecesarias, los administradores son más productivos en su trabajo.

Otro aspecto importante observado en el desarrollo de este trabajo es que, para tener éxito en el afinamiento, las reglas basadas en cotas de rendimiento no son suficientes, y es necesario complementar este conocimiento con técnicas cuantitativas y modelos.

Finalmente, como trabajo futuro, se planea la incorporación de capacidad de aprendizaje a Druida, de modo que éste se auto-afine con respecto a sus recomendaciones y los valores de la cotas de rendimiento para el afinamiento del sistema específico en que se encuentra instalado.

Asimismo, la definición automática de las clases para un MRE es fundamental para una mayor efectividad en el diagnóstico del sistema y la realización de predicciones.

AGRADECIMIENTOS

Este trabajo ha sido realizado en el contexto del proyecto Asterix del Depto. de Ciencia de la Computación de la P.U.C., con financiamiento parcial de los proyectos Fondecyt 91-0786, DIUC 90/8, y convenio DICTUC-Unisys (Chile) Corp. Se agradece especialmente la colaboración prestada por los expertos de Unisys (Chile) Corp.

REFERENCIAS

[1] Berry Robert, Hellerstein Joseph: "Experts Systems for Capacity management." 1990. CMG Transactions Summer 1990.

[2] Cady J., Howarth B. "Computer Systems Performance Management and Capacity Planning", Prentice Hall, Sydney, 1990.

[3] Casas I.,"Modelación de Sistemas Computacionales.". Apuntes de Clases. 1989.Universidad Católica de Chile, Escuela de Ingeniería, Departamento de Ciencia de la Computación.

[4] Casas Ignacio, Aurtenechea Francisco, Bustos Eduardo, Droguett Mario: "Diseño de una Herramienta para el Análisis, Supervisión y Evaluación del Rendimiento de Sistemas Unix." 1991. Universidad Católica de Chile, Escuela de Ingeniería, Departamento de Ciencia de la Computación.

[5] Dawson Jeffrey, Shubin Hal : "Interactive Expert Assistant for Computer Performance Evaluation." 1988. Conference Proceedings of CMG '88.

[6] Deese Donald : "Designing an Expert System for Computer Performance Evaluation." 1988. Conference Proceedings of CMG '88.

[7] Domansky Bernard, Soberman Sidney : "Expert Systems for Computer Performance Evaluation: On MetaRules and Knowledge Engineering." 1988. Conference Proceedings of CMG '88.

[8] Domansky Bernard: "An Expert System's Tutorial for Computer Performance evaluation." 1988. Conference Proceedings of CMG '88.

[9] Domansky Bernard, Soberman Sidney : "Expert Systems for CPE: On Evolution and Data Analysis." 1989. Conference Proceedings of CMG '89.

[10] Domansky Bernard: "A PROLOG-based Expert System for Tuning MVS/XA." 1989. Performance Evaluation Review, Febrero 1989.

[11] Domansky Bernard: "An Overview of Expert Systems." 1990. CMG Transactions Summer 1990.

[12] Droguett Mario: "Una herramienta basada en conocimiento para el afinamiento de sistemas UNIX". Tesis de Magister, Escuela de Ingeniería, Pontificia Universidad Católica de Chile, 1992.

[13] du Toit, S.H.C., Steyn A.G.W., "Graphical Exploratory Data Analysis". Springer-Verlag, New York, 1986.

[14] Roberto Holtheuer, Carlos Vizcaya "Desarrollo de una Metodología para la Evaluación y Selección de Sistemas Computacionales" Memoria de título Ingeniería, Escuela de Ingeniería, Pontificia Universidad Católica de Chile, 1989.

[15] Ishikawa, Kaoru, "Guide to Quality Control". Asian Productivity Organization, 1987.

[16] Lazowska E., et al: "Quantitative System Performance, Computer System Analysis Using Queueing Network Models" Prentice-Hall New Jersey 1984.

[17] Lee Paul, Farrel Brian, Dronamraju Krishna: "Performance Measurement and Evaluation in Unix System V." 1989. Conference Proceedings of CMG '89.

[18] Madhar S. Phadke: "Quality Engineering using robust design", Prentice-Hall, New Jersey, 1989.

[19] Ovalle F. "Modelación y Evaluación del desempeño de un Sistema Computacional Centralizado". Memoria de Título, Departamento de Computación, Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Abril de 1988.

[20] Ramamurthy G. : "A Capacity Planning and Configuration Modeling Methodology for Unix Systems" 1989. Conference Proceedings of CMG '89.

[21] Samadi Behrokh: "TUNEX: A Knowledge-Based System for Performance Tuning of the Unix Operating System." 1989. IEEE Transactions on Software Engineering, Julio 1989.

[22] Stroustrup Bjarne: "The C++ Programming Language" Addison-Wesley 1985.

[23] Unisys Corp. "Unix System V : Administrator Guide" Versión de sistema 3.0.

[24] Zimmer Harry : "Rules of Thumb '90." 1990. CMG Transactions Spring 1990.

Reconocimiento de Palabras Aisladas utilizando Modelos Ocultos de Markov* .

Maria Teresa Escrig Monferrer.[1]

Francisco Casacuberta Nolla.[2]

[1]*Departamento de Matemáticas e Informática.*
Universidad Jaume I de Castellón.
Campus Penyeta Roja. 12071 Castellón, España.

[2]*Departamento de Sistemas Informáticos y Computación.*
Universidad Politécnica de Valencia.
Valencia, España.

Key words: Automatic Speech Recognition, Isolated Word Recognition, Hidden Markov Models, Speaker Independent Recognition.

RESUMEN.

Actualmente son necesarias interfaces en las cuales la comunicación entre el ser humano y las computadoras sea eficiente, natural y que no requiera un gran esfuerzo de aprendizaje.En ello trabaja la línea de investigación "Reconocimiento Automático del Habla". En este trabajo se describe un sistema para el **Reconocimiento de Palabras Aisladas (RPA)** para vocabularios reducidos, independiente del locutor.y se muestran los resultados de su implementación en una estación de trabajo HP-9000, sobre la que el sistema funciona en tiempo real.

A partir de la señal vocal (en nuestro caso una palabra aislada del vocabulario), se obtiene una representación paramétrica de dicha señal en la etapa denominada **preproceso**, en el cual se realiza un tratamiento mediante **Banco de Filtros**, un cálculo del **cepstrum** y un **etiquetado**, que reducirá la cantidad de información con la que tratar y enfatizará las características más relevantes de la señal.

Señales así parametrizadas serán utilizadas para el aprendizaje de un **Modelo de Markov Oculto (MMO)** para cada palabra del vocabulario. Para el reconocimiento se tomarán muestras igualmente parametrizadas y se calculará la probabilidad de generar dicha muestra por cada uno de los modelos, eligiéndose como palabra aquella cuyo modelo de la probabilidad más alta.

El sistema permite, además, la posibilidad de generación de nuevas tareas, aprendiendo los modelos de cualquier otro vocabulario.

ABSTRACT.

Nowardays it is needed interfaces in which human-computer comunication is efficient, natural and that don't require a large effort of training. The line of investigation"Speech Automatic Recognition" is working in

* Este proyecto ha sido parcialmente subvencionado por la Comisión Internacional de Ciencia y Tecnología (CICYT): "Construcción de Sistemas de Reconocimiento de Habla" TIC 448/89.

this subjet. In this work we show a Speaker-Independent, Isolated Word Recognizer for small vocabularies, and we show the implementation results on a workstation HP-9000, in real time.

We obtain a parametric representation from a speech signal (in our case an isolated word of the vocabulary) in a preprocessing stage in which a filter-bank treatment, a cepstrum calculation and a labelling are made. This will reduce the amount of information to be treated and will emphasize the most outstanding signal features.

This parametric signal will be used for training a distinct Hidden Markov Model (HMM) for each word of the vocabulary. Recognition consists of computing the probability of generating the test word with each model and choosing the word whose model gives the highest probability.

This system allows us to generate new tasks to train models of other vocabulary.

1. INTRODUCCION.

El área de investigación en la cual está enmarcado este trabajo, Reconocimiento Automático del Habla (RAH), apunta hacia un estudio obligatorio en nuestra sociedad actual, el de las interfaces que faciliten la comunicación entre el ser humano y sus computadoras de forma eficiente, natural y sin requerir gran esfuerzo de aprendizaje.Entre la distintas disciplinas que confluyen en el RAH destacan la Teoría de la Señal, el Reconocimiento de Formas y la Inteligencia Artificial.

Debido a la enorme dificultad que entraña la comunicación oral, aún no ha sido posible diseñar un sistema de RAH general, que interprete cualquier discurso de cualquier locutor. Para simplificar el problema general hasta llevarlo a planteaminetos abordables, se introducen restricciones que hacen que los sistemas resultantes queden especializados en determinados aspectos del habla, por ejemplo Reconocimiento de Palabras Aisladas [3] o Reconocimiento del Discurso Continuo.

En este trabajo se describe un sistema para Reconocimiento de Palabras Aisladas (RPA) independiente del locutor y utilizando unos vocabularios reducidos.

A partir de la señal vocal (en nuestro caso, una palabra aislada del vocabulario), todo sistema de RAH debe obtener una representación paramétrica de dicha señal en una etapa denominada preproceso [9] [10], que reducirá la cantidad de información con la que tratar y enfatizará las características más relevantes de la señal.

Los vectores de parámetros acústicos serán clasificados en categorias microfonéticas en un proceso denominado etiquetado. La señal resultado de ese proceso servirá para el aprendizaje de una estructura o modelo que representará a cada palabra del vocabulario.

El criterio seguido en la etapa de reconocimiento, para determinar a qué palabra del vocabulario corresponde la palabra pronunciada, es el de máxima probabilidad [2] [4].

La estructura que representa cada palabra del vocabulario es un Modelo Oculto de Markov (MMO) que es la base de una técnica probabilística [3] [6] [10] [13].

2. ESTRUCTURA DEL SISTEMA.

En la figura 1 podemos ver un esquema del sistema que constituye nuestro reconocedor de palabras aisladas [6].

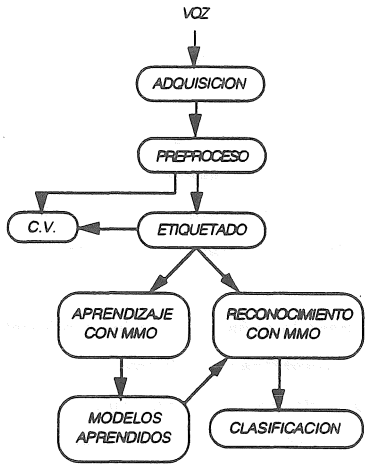


Figura 1. Sistema propuesto para nuestro Reconocedor de Palabras Aisladas.

La señal vocal podrá ser utilizada para el aprendizaje de un modelo de Markov que la representa, después del paso por las fases de adquisición, parametrización y etiquetado, denominadas en conjunto **preproceso** [6].

En la fase de **adquisición** [2], las ondas de presión producidas por el aparato fonador humano, son captadas por un micrófono que convierte esa onda sonora de presión en señal eléctrica. Posteriormente es amplificada para llevar la débil señal eléctrica obtenida hasta niveles manejables y después filtrada.

La señal analógica captada y amplificada, es digitalizada por un conversor analógico-digital; muestreando la señal analógica (midiendo la amplitud de la señal cada cierto tiempo) a 8555 Hz. y cuantificando la señal muestreada (pasando a una serie de valores numéricos) a 12 bits. Esta señal muestreada debe cumplir el teorema de muestreo o de Nyquist que indica que la máxima frecuencia del espectro no nulo de la señal a muestrear debe ser inferior a la mitad de la frecuencia de muestreo, de lo contrario se producen solapamientos de los espectros sucesivos que impiden reconstruir la señal original (efecto "aliasing") [11].

La siguiente fase por la que pasa la señal vocal es la de **parametrización** que servirá para eliminar la información redundante y enfatizar aquella que es relevante [2]. La parametrización consta de una multiplicación por ventanas de 256 puntos por la señal vocal de forma solapada. La ventana utilizada es la de HAMMING de 30 ms., por ser la más adecuada cuando la transformación aplicada a la señal vocal va dirigida a un análisis frecuencial de la señal [2], con un desplazamiento de 15 ms.

Después se realiza el cálculo de la Transformada de Fourier Dependiente del Tiempo que es procesada con la **Transformada Rápida de Fourier (FFT)** [7] [11].

Posteriormente se realiza un cálculo mediante **Banco de Filtros** de N filtros pasabanda, con sus intervalos frecuenciales distribuidos según unas escalas, que representan la división natural de las frecuencias audibles por el ser humano (bandas críticas) y su unidad es el "mel". La escala utilizada es la ESCALA DE MEL de 18 filtros que modeliza la respuesta de la membrana basilar del oido y cuyos resultados han sido comprobados en la práctica [2] [3].

Después del tratamiento de la señal mediante un banco de filtros, se realiza el cálculo de **coeficientes cepstrales** mediante la transformada inversa del coseno del logaritmo de los módulos [11].

La siguiente fase por la que pasa la señal vocal es la de **etiquetado**. Previamente a esta fase se ha tenido que efectuar una **Cuantificación Vectorial** (CV), técnica de Reconocimiento de Formas basada en una técnica de agrupamiento para el aprendizaje no supervisado ("Clustering"), que permitirá reducir la cantidad de información necesaria para representar la señal vocal. Mediante la CV creamos un conjunto de vectores prototipo ("codebook") a partir de unos vectores muestra, que definen un conjunto de clases. A cada clase se le asigna una etiqueta que la identifica, en principio de forma aleatoria [1] [5] [12].

Una vez definidas las clases y los prototipos representantes de cada clase, en la fase de etiquetado se asignará una etiqueta a cada vector de parámetros obtenido en la fase de parametrización utilizando la técnica de los k-vecinos más próximos, con k=3. La complejidad de este algoritmo es proporcional al número de prototipos por cada vector de parámetros.

Señales procesadas según lo anteriormente expuesto servirán, en una fase de entrenamiento, para el aprendizaje de los **Modelos de Markov Ocultos (MMO)** que representen cada una de las palabras del vocabulario. En una etapa de reconocimiento, se elegirá como palabra pronunciada aquella cuyo modelo ha dado la probabilidad más alta de generación de la muestra a reconocer.

3. MODELOS OCULTOS DE MARKOV.

Un MMO consta de una componente estructural, que viene dada por un conjunto de estados finito, y otra probabilística, formada por conjuntos de funciones de distribución de probabilidades [4] [10].

Transición entre estados $A = \{a_{ij}\} / a_{ij} \in [0,1]$; $1 \le i,j \le N$, que cumple que $\Sigma a_{ij} = 1$
Emisión de símbolos $B = \{b_i(k)\} / b_i(k) \in [0,1]$; $1 \le i \le N$; $1 \le k \le M$, que cumple que $\Sigma b_i(k) = 1$
Y distribución de estados iniciales $\pi = \{\pi_i\}$, que cumple que $\Sigma \pi_i = 1$

Sin embargo, debido a los buenos resultados obtenidos empíricamente [10] se utiliza una estructura de MMO con unas restricciones, dando lugar a los Modelos de Markov Ocultos de Izquierda a Derecha (MMOID) (figura 2), caracterizado por las propiedades siguientes:

1- $a_{ij} = 0$ si $j<i$.

2- $\pi_1 = 1$ y $\pi_j = 0$ para todo $j<>1$.

3- q_N es un estado final o absorbente.

A partir de los resultados experimentales obtenidos en [10] elegimos para este trabajo Modelos de Markov de Izquierda a Derecha (MMOID), sin saltos entre estados no continuos y con 10 estados, aunque, como veremos en el apartado siguiente se realizaron algunos experimentos con modelos de 20 estados.

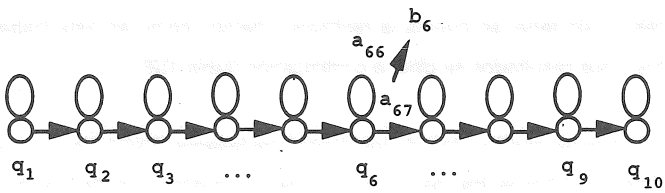


Figura 2. Modelo de Markov de Izquierda a Derecha de 10 estados y sin saltos entre estados no contiguos.

Al utilizar los MMOID se plantearon una serie de problemas que son expuestos en los siguientes párrafos.

Dada una secuencia de observaciones O, será reconocida como aquella palabra perteneciente al vocabulario ($w^* \in V$) cuyo MMO M dé la probabilidad más alta de generar la muestra [4] [6] [10].

$$w^* = \underset{1 \le v \le V}{\operatorname{argmax}} P_v(M/O)$$

El cálculo de la probabilidad de que un modelo genere una muestra O se realiza mediante los algoritmos "Hacia Delante" o "Hacia Atrás" [6] [10].

El problema de la precisión o "underflow" (probabilidades tendentes a 0) que es clásicamente resuelto introduciendo un factor de escalado [10], se soluciona en este trabajo utilizando una representación logarítmica de las probabilidades [6] [13]. Se pasa de trabajar con números reales a trabajar con números enteros. Además el cálculo de logaritmos simplifica algunas operaciones como son la multiplicación (suma de los logaritmos) y la suma que, aunque es más complejo, puede tabularse (en una tabla cuyo tamaño depende de la base logarítmica), de manera que el logaritmo de una suma se reduce a una suma, una resta, una comparación y una búsqueda en tabla. A pesar de que con los logaritmos se introducen errores, se demuestra empíricamente [13] que produce idénticos resultados a los obtenidos mediante la técnica del escalado.

La sustitución del escalado por el cálculo del logaritmo de las probabilidades supuso un ahorro sustancial del tiempo requerido para el aprendizaje y el reconocimiento utilizando los MMOID [6] [13].

4. EXPERIMENTOS Y RESULTADOS.

Los experimentos realizados con nuestro reconocedor tuvieron lugar en un ambiente ruidoso de un laboratorio de trabajo, lo cual, sin lugar a dudas, influyó en que los resultados obtenidos sean ligeramente peores a aquellos obtenidos con muestras adquiridas sin ruidos y comprobadas con un módulo de reconocimiento no integrado con el sistema de adquisición/preproceso y sin tener en cuenta la restricción tiempo como en este trabajo. Algunos de esos experimentos y sus resultados se citan a continuación (tabla 1)[6].

- Reconocimiento, en sistema multilocutor, de los dígitos castellanos (**DIG1**), en el cual, para 300 muestras de aprendizaje y 210 muestras para reconocimiento, se obtuvo un 94'3 % de aciertos, en un tiempo medio de reconocimiento de 1 s. 85 cs.

- Reconocimiento, en sistema multilocutor, de los dígitos castellanos, pero utilizando un modelo por cada dígito y cada sexo (**DIG2**), con un número similar de muestras para aprendizaje y reconocimiento, se obtuvo un 93'5 % de aciertos en reconocimiento para hombres y un 95' 7% para las mujeres. Lo cual indica que, a pesar del incremento del espacio y el tiempo necesario que supone aprender modelos por separado para los hombres y las mujeres los resultados son similares.

Tabla 1. Tabla de resultados obtenidos para diversos experimentos (ver texto), en los que se indica el número de muestras utilizado para el aprendizaje y el test, los tiempos de aprendizaje y de test medios, así como las tasas de muestras reconocidas correctamente.

	APRENDIZAJE		RECONOCIMIENTO		
	MUESTRAS	TIEMPO	MUESTRAS	TIEMPO	ACIERTOS
DIG1	300 (3h, 3m)	5m. 26s.	210 (1h, 3m)	1s. 85cs.	94'3%
DIG2	250 300	5m. 26s.	140 140	1s. 85cs.	93'5% 95'7%
DIG3	200 100	5m. 26s.	100 190	1s. 85cs.	80% 80%
NOMPER30	900 (3h, 3m)	19m. 28s.	250 (1h, 2m)	2s. 71cs.	87'2%
CIUD_20E	100	9m. 3s.	80	4s. 26cs.	99'5%

- Reconocimiento, en sistema multilocutor, de los dígitos castellanos separando modelos para hombres y mujeres, pero con un aprendizaje común inicial para ambos (**DIG3**); con un número similar de muestras, el porcentaje de aciertos en el reconocimiento disminuye hasta un 80 % tanto para hombres como para mujeres, debido a que los modelos están aprendidos con demasiadas iteraciones y llegan a "especializarse" en las muestras aprendidas.

- Los experimentos realizados con vocabularios más extensos (30 nombres de personas) **(NOMPER30)** y, con un número similar de muestras de aprendizaje, daban aún resultados peores, demostrando que se necesitan muchas muestras de aprendizaje para entrenar los modelos de vocabularios extensos y palabras parecidas.

- Se realizaron varios experimentos en un sistema monolocutor, con un vocabulario de 20 nombres de ciudades, de los que podemos extraer las siguientes conclusiones. El porcentaje de aciertos aumenta tal como vamos aprendiendo más los modelos hasta un cierto punto en el cual disminuye, por el problema antes mencionado. Si se utiliza un codebook propio para la aplicación la tasa de reconocimiento aumenta significativamente. El mejor resultado se obtuvo aprendiendo los modelos sobre un modelo de 20 estados sin saltos entre estados no contiguos **(CIUD_20E)** en vez de los modelos de 10 estados utilizados en los experimentos anteriores , 99'5%. Sin embargo, el tiempo medio de reconocimiento, aumenta en este caso significativamente.

5. CONCLUSIONES.

La construcción de modelos separados para hombres y mujeres no producen una mejora en el porcentaje de aciertos en el reconocimiento respecto a los que no hacen esa disticción. Para vocabularios relativamente extensos y con palabras parecidas, se necesitan bastantes muestras de aprendizaje para entrenar los modelos y obtener porcentajes de aciertos aceptables. En un sistema monolocutor (y con un codebook genérico) el mejor resultado se obtiene aprendiendo los modelos en dos fases, utilizando 5 muestras de cada palabra cada vez y con 10 iteraciones en el algoritmo de aprendizaje. Si los modelos están más entrenados la tasa de reconocimiento decrece. Para el mismo sistema obtenemos mejores resultados si modificamos la topología de los MMOID de 10 a 20 estados, aunque el tiempo de respuesta al usuario en este caso sea inaceptable.

Posibles mejoras al reconocedor construido serían la aplicación del algoritmo de Viterbi para el reconocimiento, lo cual podría mejorar el tiempo de respuesta; realizar una detección de bordes más fina; y distinguir las palabras pronunciadas que no pertenecen al vocabulario.

6. AGRADECIMIENTOS.

Deseo expresar mi más sincero agradecimiento a mi director de proyecto y al catedrático Enrique Vidal Ruiz, miembros del grupo de investigación "Reconocimiento Automático del Habla" de la Universidad Politécnica de Valencia, por su inestimable apoyo.

7. REFERENCIAS.

[1] Gabriela Andreu, Enrique Vidal y Francisco Casacuberta. **"An Empirical Evaluation of feature Maps and other Clustering Tecnhniques for frame labeling of speech"**. Elsevier Science Publisher B.V., 1990.

[2] Jose miguel Benedí Ruiz. **"Estudio de un Sistema de Reconocimiento Automático del Habla: TABARCA"**. Tesis doctoral. Universidad Politécnica de Valencia. 1988.

[3] Enrique Vidal Ruiz y Francisco Casacuberta Nolla. **"Reconocimiento Automático del Habla"**. MARCOMBO. BARCELONA. 1989.

[4] Francisco Casacuberta Nolla. **"Modelos de Markov Ocultos y Reconocimiento de Palabras Aisladas"**. DSIC. Universidad Politécnica de Valencia. Febrero 1991.

[5] R.O. Duda, P.E. Hart. **"Pattern Classification and Scene Analisys"**. John Wiley and Sons. 1973.

[6] María Teresa Escrig Monferrer. **"Un Reconocedor de Palabras Aisladas en tiempo real, para una estación de trabajo HP-9000 utilizando Modelos Ocultos de Markov"**. Proyecto Fin de Carrera. Universidad Politécnica de Valencia. 1991.

[7] F.J. Harris. **"On the use of windows for Harmonic Analisys with the Discrete Fourier Transform"**. Proc. IEEE, Vol 66, No 1, pp 51-84. 1978.

[8] S. Levinson. **"A Unified Theory of Composite Pattern Analisys for Automatic Speech Recongnition"**. En "Computer Speech Processing". F. Fallside y W.A. Woods. Prentice-Hall. 1985.

[9] Andrés Marzal Varó. **"Nuevos Métodos de Segmentación para la Decodificación Acústico-Fonética"**. Proyecto Fin de Carrera. Universidad Politécnica de Valencia. 1990.

[10] Begoña Más Valor. **"Reconocimiento de Palabras Aisladas y Decodificación Acústico-Fonética utilizando Modelos Ocultos de Markov"**. Proyecto Fin de Carrera. Universidad Politécnica de Valencia. 1989.

[11] L.R. Rabiner, R.W. Schafer. **"Digital Processing of Speech Signals"**. Prentice-Hall. 1978.

[12] L.R. Rabiner, S.E. Levison, M.M. Sondhi. **"On the aplication of vector Quantization and Hidden Markov Models to Speaker-independent, Isolated Word Recognition"**. The Beel System Tech. Journal. Vol. 62, No 4, pp 1075-1104. 1983.

[13] Luís Sanchez. **"Decodificación Acústico-Fonética utilizando Modelos Ocultos de Markov"**. Proyecto Fin de Carrera. Enero 1992.

Adaptive Parallel Sorting

Vladimir Estivill-Castro* Derick Wood†

March 4, 1992

Abstract

The capacity of an algorithm to modify its behavior to the size, difficulty or structure of the problem instance has been named adaptivity. The literature on parallel algorithms qualifies as adaptive those algorithms that work with a fixed amount of processors on input of arbitrary size while the literature on sorting algorithms qualifies as adaptive those algorithms that perform work proportional to the amount of disorder in the input. We combine these two concepts and present the first two parallel sorting algorithms that

- are designed for a fixed number of processors and reconfigure according to the size of the input
- adapt to the amount of disorder in the input, and
- are cost optimal.

1 Introduction

Over the last few years, it has become increasingly evident that parallelism will provide an opportunity for faster, more efficient and more cost-effective computation. Because of its simplicity and universality, the *P-RAM* (Parallel Random Access Machine) model has resulted in a theoretically convenient parallel model of computation [11,18]. The *P-RAM* offers a flexible starting point for the designer of parallel algorithms. In most situations, however, when a problem involving n data items is to be solved in parallel, it is unrealistic on the part of the algorithm designer to assume that the number of available processors is bigger than or equal to n, especially when n is extremely large. Therefore, algorithms that *adapt* to a sublinear number of available processors and whose running time is a function of n as well as of the number of processors are desirable [2, page 159]. For the problem of sorting n elements, another property of the algorithm is desirable. The algorithms should *adapt* to the amount of disorder in the data; that is, its *cost* (the product of the number of processors and its running time) is a smooth non-decreasing function of the size and the disorder in

*Department of Computer Science, York University, North York, Ontario M3J 1P3, Canada.

†Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada

the input. Adaptive sorting algorithms for sequential computation are well-studied [9,10,17]. This paper presents the first parallel sorting algorithms that:

- are designed for a fixed number of procesors p and modify their behavior and running time according to the size of the input, (also referred as *self-reconfiguring* [4])
- adapt to the amount of disorder in the input,
- their worst-case running time varies inversely with p and is significantly smaller than the best sequential algorithm, and
- the cost of the parallel algorithm is optimal.

Our model of parallel computation consists of a *P-RAM* where there are $p \geq 1$ processors working synchronously and communicating though a random access memory. Each processor is a uniform-cost Random Access Machine (*RAM*) with usual operations and instructions (i.e. the cost of comparisons and arithmetic operations is constant). The processors are indexed by the natural numbers $P_0, P_1, \ldots, P_{p-1}$. During a given time unit a selected number of processors are active and execute the same instruction, each on a different data set; the remaining processors are inactive. Since the processors execute the same program such a model is also called a single-instruction, multiple-data-stream (*SIMD*) model [3].

In one step each processor can access (either reading from it or writing to it) one memory location. In the literature, *P-RAM* models differ in regard to simultaneous access of the same memory location by more than one processor. We use the following convention:

- *CRCW-P-RAM*: two processors are allowed to simultaneously read from or write into the same memory cell.
- *CREW-P-RAM*: two processors are allowed to simultaneously read from the same memory cell but concurrent writing in not allowed.
- *EREW-P-RAM*: no two processors are allowed to simultaneously read from or write into the same memory cell.

For the *CREW-P-RAM* and *EREW-P-RAM* models it is assumed that there is an initial threshold time of $\log p$ to activate p processors at the beginning of every computation [11,18].

To evaluate parallel algorithms we use the following well-established metrics [2,3]. Let X be a sequence of n elements from a total order, the parallel running time of a sorting algorithm A is the number of steps performed by A to sort X and is denoted by $T_A(X)$. The cost of a parallel sorting algorithm A is defined as the produc $p \cdot T_A(X)$ and is denoted by $C_A(X)$. A parallel algorithm is said to be *cost-optimal* if, for all X, $C_A(X)$ is within a constant factor of the lower bound. In this paper we adopt the common convention that the running time of the optimal sequential algorithm is a lower bound for the cost of a parallel algorithm. Moreover, we also adopt the convention that,

for any algorithm A on a *CRCW-P-RAM* with p processors the cost is at least p, since A will take one time step to activate the processors. Similarly, for any algorithm A on a *CREW-P-RAM* or *EREW-P-RAM* with p processors, the cost of A is at least $p \log p$ since A will take $\log p$ time to activate the processors (or broadcast n, the size of the input).

Our goal is to obtain a cost-optimal parallel sorting algorithms for all values of p and $n \geq p$.

In a comparison-based model of computation $\Omega(n \log n)$ comparisons are necessary in the worst-case to sort a sequence of length n. Frequently, the input is nearly-sorted and it can be sorted using less comparisons. Traditional sorting algorithms like *Quicksort* or *Mergesort* are oblivious to the existing order in the sequence (they even sort again an already sorted sequence). Algorithms that profit from the order in the input sequence are said to be *adaptive* [16, page 224]. The adaptive behavior of a sorting algorithm A is demonstrated by showing that the time required by A to sort a sequence X is a non-decreasing function of the size of X and the disorder in X. The following are common measures of disorder. The largest distance, $Dis(X)$, determined by an inversion in $X = \langle x_1, \ldots, x_n \rangle$, where (i, j) is an *inversion* if $i < j$ and $x_i > x_j$. The largest distance, $Max(X)$, an element must travel to reach its sorted position. The number, $Runs(X)$, of ascending runs.

A sequential sorting algorithm is optimally adaptive with respect to a measure of disorder M (or M-optimal) if it achieves, within a constant factor, maximal adaptivity for every input sequence [14].

Sequential adaptive sorting has been investigated extensively [9,10,17]; however, the literature on adaptive sorting for parallel computation consists of only a few special cases. Altman and Igarashi [6] present parallel algorithms requiring n processors that adapt to a previously known Dis value. Altman and Chlebus [5] present an algorithm that is cost-optimal for Dis on a *CRCW-P-RAM* with n processors (note that the number of processors must equal the size of the input). Carlsson and Chen [8] present an algorithm that is cost-optimal for $Runs$ on a *EREW-P-RAM* with $n/\log n$ processors. Levcopoulos and Petersson [13] claim to obtain a cost-optimal algorithm for Rem. However, when sorting a sequence X with n elements the algorithm uses $n/\log n$ processors during two phases that last $O(\log n)$ time and $O([n + Rem(X) \log Rem(X)]/logn)$ processors during one phase that last $O(\log n)$. The cost of their algorithm is at least $\Omega(n \log n)$ and not $O([n + Rem(X) \log Rem(X)]/logn)$ as they claim. Levcopoulos and Petersson [12] also propose a different model of computation. At any step the parallel algorithm may decide to declare several processors as idle and not be charged for them. The algorithm has full power to request more processors and they are supplied with no delay. Thus, the cost of the algorithm is redefined [12] as the sum over all time steps t_i of the number of processors active at step t_i. In this model, a parallel sorting algorithm is described. This algorithm estimates the number of inversions, (requesting a convenient number of processors in several phases), and sorts optimally with respect to Inv. Note that the *P-RAM* differs drastically from this model in that the number of processors is fixed through the computation and the algorithm can not avoid to be charged for those processors that declares idle. McGlinn [15] has studied adaptive sorting empirically.

Adaptive Parallel Merge: Merges s sorted sequences using $p = n^{1-\epsilon}$ processors.

for $i := 1$ to $\log s$
- Form $\lfloor s/2^i \rfloor$ pairs of sequences.
- Distribute the processors among the pairs and assign $\max\{1, (l'_i + l'_j)p/n\}$ processors to each pairs of sequences of lengths l'_i and l'_j.
- In parallel, merge each pair into a sorted sequence using Bilardi and Nicolau's algorithm.

Figure 1: Pseudo-code for *Adaptive Parallel Merge*

In this paper we present adaptive parallel algorithms that sort optimally with respect to a measure of disorder and use a fixed number of processors. In Section 2 we introduce *Parallel Mod Sort*, an adaptive parallel algorithm that is cost-optimal with respect to the measure Dis, and the measure Max, for all positive values of n and $p = n^{1-\epsilon}$ ($0 < \epsilon < 1$). Moreover, *Parallel Mod Sort* is cost-optimal for all variants of the *P-RAM*. In Section 3 we introduce *Parallel Block Sort*, an adaptive parallel algorithm that is cost-optimal with respect to the measure *Runs*, for all positive values of n and $p = n^{1-\epsilon}$ ($0 < \epsilon < 1$). Moreover, *Parallel Mod Sort* is cost-optimal for all variants of the *P-RAM*.

2 A cost-optimal algorithm for Dis and Max

In this section we present a parallel sorting algorithm that is cost-optimal with respect to Dis and adapts to the size of the input. The algorithm can be implemented in all *P-RAM* models. In order to present our sorting algorithm we must present *Adaptive Parallel Merge* (see Figure 1), an adaptive algorithm for merging $s(\leq p)$ sorted sequences, using p processors in a *EREW-P-RAM*. The algorithm is a generalization of Bilardi and Nicolau's [7] algorithm for merging two sorted sequences of total length n in $O(n/p + \log n)$ time using p processors.

Lemma 2.1 *Let S be a set of s sorted sequnces of lengths $l_1, l_2, \ldots, l_s$ (with $\sum_{i=1}^{s} l_i = n$). Assume the values s and l_i, for $i = 1, \ldots, s$ are known to all processors. Adaptive-Merge uses $p = n^{1-\epsilon}$ ($0 < \epsilon < 1$) processors to merge these s sorted sequences in $O([n/p]\log(s+1))$ time on a P-RAM.*

Proof: The algorithm is the same for all variants of the *P-RAM*. Each step is an application of Bilardi and Nicolau merging algorithm, thus each step takes

$$O\left(\frac{l'_i + l'_j}{(l'_i + l'_j)n^{-\epsilon}} + \log(l'_i + l'_j)\right)$$

time, for some l'_i, l'_j with $l'_i + l'_j \leq n$. Thus, each step requires $O(n^\epsilon + \log n) = O(n^\epsilon)$ time. Since the total number of step is $\lceil \log s \rceil$, the lemma follows. □

We use the following notation. Given an array $A = A[0, \ldots, n-1]$, we denote the subarray of A consisting of all elements in position l to position r by $A[l, \ldots, r]$. A subsequence of A is the sequence of elements $A[i(0)], A[i(1)], \ldots, A[i(s)]$ where $i : \{1, 2, \ldots, s\} \rightarrow \{0, \ldots, n-1\}$ is injective and monotonically increasing. For positive m, we denote by $\{A_k\}_m$ the subsequence determined by $i(r) = m \cdot r + k$, for $k = \{0, \ldots, n-1\}$. For example, for $m = 2$, $\{A_0\}_2$ denotes the subsequence of elements in even positions. We present now the sorting algorithm.

Parallel Mod Sort: Sorts an array $A[0, \ldots, n-1]$ with processors $P_0, \ldots, P_{p-1}$.

1.- *Divide the input into p subarrays.* Let $p-1$ of these subarrays consist of $\lceil n/p \rceil$ elements and the remaining subarray contains $n + 1 - (p-1)\lceil n/p \rceil (\leq \lceil n/p \rceil)$ elements. Assign processor P_i to the subarray $A[i\lceil n/p \rceil, \ldots, i\lceil n/p \rceil + \lceil n/p \rceil - 1]$, for $i = 0, \ldots, p-2$, and assign P_{p-1} to the subarray $A[(p-1)\lceil n/p \rceil, \ldots, n]$.

2.- *Sort each subarray.* In parallel, each processor P_i, for $i = 0, \ldots, p-1$, sorts its subarray using a sequential *Dis*-optimal sorting algorithm.

3.- *Test for sortedness.* In parallel, processor P_i, for $i = 0, \ldots, p-2$, checks if $A[i\lceil n/p \rceil + \lceil n/p \rceil - 1] > A[(i+1)\lceil n/p \rceil]$ and the "or" of these boolean values is computed. If A is sorted, terminate.

4.- *Divide the input into $p-1$ subarrays shifted* $\lceil n/2p \rceil$. Let these $p-1$ subarrays consist of $\lceil n/p \rceil$ elements. Assign processor P_i to the subarray $A[\lceil n/2p \rceil + i\lceil n/p \rceil, \ldots, \lceil n/2p \rceil + (i+1)\lceil n/p \rceil - 1]$, for $i = 0, \ldots, p-2$.

5.- *Sort each subarray.* In parallel, each processor P_i, for $i = 0, \ldots, p-2$, sorts its subarray using a sequential *Dis*-optimal algorithm.

6.- *Test for sortedness.* In parallel, processor P_i, for $i = 0, \ldots, p-2$, checks if $A[\lceil n/2p \rceil + (i+1)\lceil n/p \rceil - 1] > A[\lceil n/2p \rceil + (i-1)\lceil n/p \rceil]$ and the "or" of these boolean values is

computed. If A is sorted, terminate.

7.- *Compute k an estimate for $Dis(A)$.* Let R be the subsequence of A consisting of the elements in the subsequences $\{A_{\lceil n/2p\rceil+1}\}_{\lceil n/p\rceil}$ and $\{A_{\lceil n/2p\rceil}\}_{\lceil n/p\rceil}$. Use Altamn and Chlebus parallel algorithm to compute $Dis(R)$. Let $k = \lceil Dis(R)/2\rceil \lceil n/p\rceil$.

8.- *Merge subsequences formed by elements $k+1$ positions apart.*

8.1 If $p \leq k+1$ then,

8.1.1 For $i = 0, \ldots, k$, let $A_i = \{A_i\}_{k+1}$. Assign

$$A_{j\lceil (k+1)/p\rceil}, A_{j\lceil (k+1)/p\rceil+1}, \cdots A_{(j+1)\lceil (k+1)/p\rceil-1}$$

to processor P_j for $j = 0, \ldots, p-2$ and $A_{(p-1)\lceil (k+1)/p\rceil}, \ldots, A_{k+1}$ to processor P_{p-1}. In parallel, each processor merges its subsequences into a sorted subsequence.

8.1.2 Use *Adaptive Parallel Merge* to merge the resulting p subsequences, all of length no more than $\lceil n/p\rceil$ using p processors.

8.2 If $p > k+1$, use *Adaptive Parallel Merge* to merge $\{A_0\}_{k+1}, \ldots, \{A_k\}_{k+1}$ (each of length no more than $\lceil n/(k+1)\rceil$) using p processors.

The merit of the algorithm is expressed in the following theorem.

Theorem 2.2 *Let X be a sequence of n elements from a total order.*

- *In a CRCW-P-RAM with $p = n^{1-\epsilon}$ processors $(0 < \epsilon < 1)$ Parallel Mod Sort sorts X in $O((n/p)[1+\log(Dis(X)+1)])$ time. The cost of Parallel Mod Sort is $O(n[1+\log(Dis(X)+1)]+p)$ units which is Max and Dis-optimal.*
- *In a EREW-P-RAM or CREW-P-RAM with $p = n^{1-\epsilon}$ processors $(0 < \epsilon < 1)$ Parallel Mod Sort sorts X in $O((n/p)[1+\log(Dis(X)+1)]+\log p)$ time. The cost of Parallel Mod Sort is $O(n[1+\log(Dis(X)+1)]+p\log p)$ units which is Max and Dis-optimal.*

Proof: Let X be a sequence of length n. The correctness of the algorithm follows from the fact that it terminates at steps 3 or 6 when it has verified that the array is sorted. If the algorithm reaches step 7, it can be verified that

$$\left\lceil \frac{Dis(R)+2}{2} \right\rceil \left\lceil \frac{n}{p} \right\rceil \geq Dis(X) \geq \left\lfloor \frac{Dis(R)-1}{2} \right\rfloor \left\lceil \frac{n}{p} \right\rceil .$$

Thus, the algorithm terminates at step 8 when it merges a partition of the array consisting of $k+1$ sorted subsequences.

We now analyze the time used by the algorithm. Clearly, steps 1, 3, 4, and 6 require constant time in a *CRCW-P-RAM* while they require $O(\log p)$ time in a *EREW-P-RAM* or *CREW-P-RAM*. Since a sequential *Dis*-optimal algorithm is used in steps 2 and 5, the time used in these steps is

$$O((n/p)[1+\log(1+\min\{n/p, Dis(X)\})]) = O((n/p)[1+\log[1+Dis(X)]).$$

Thus, if the algorithm finishes at steps 3 or 5, the theorem follows. It can be verified that Altman and Chlebus's algorithm for computing *Dis* can be extended to an algorithm with exclusive writing provided that the paralllel computation of prefixes for associative operations is performed in $\log p$ time. Thus step 7 can be performed in

$$O(\log[Dis(R)+1]) = O(\log[Dis(X)+1])$$

time on a *CRCW-P-RAM* and in

$$O(\log[Dis(R)+1]+\log p) = O(\log[Dis(X)+1]+\log p)$$

time on a *EREW-P-RAM* or on a *CREW-P-RAM*.

If $p < k+1$, then $p \leq Dis(X)$ and the algorithm performs step 8.1. During step 8.1.1 each processor merges $\lceil (k+1)/p \rceil$ sequences each of length no more than $\lceil n/(k+1) \rceil$. Thus, this takes $O((n/p)\log\lceil (k+1)/p \rceil) = O((n/p)\log[1+Dis(X)])$ time. Now, by Lemma 2.1, step 8.1.2 takes $O((n/p)\log(p+1)) = O((n/p)\log[Dis(X)+1])$ time (*logp* additional time is required on a *CREW-P-RAM* or *EREW-P-RAM* to broadcast the parameters for the merge).

If $p \geq k+1$, by Lemma 2.1, step 8.2 on a *CRCW-P-RAM* takes $O((n/p)\log(k+1))$ for $p = n^{1-\epsilon}$, $0 < \epsilon < 1$. Similarly, on a *EREW-P-RAM* and on an *CREW-P-RAM* this step last $O((n/p)\log(k+1)+\log p)$ time.

From this analysis and the fact that a sequential *Dis*-optimal algorithm sorts in $\Omega(n[1+\log(Dis(X)+1)])$ [9] the theorem follows. □

3 A cost-optimal algorithm for *Runs*

The ideas of the previous section allow us to easily generalize Carlsson and Chen's algorithm [8] to a parallel sorting algorithm that is cost-optimal with respect to *Runs* and adapts to the size of the input. The algorithm can be implemented in all *P-RAM* models.

Parallel Block Sort. Sorts and array $A[0,\ldots,n-1]$ with processors $P_0,\ldots,P_{p-1}$.

1.- *Divide the input into p subarrays.* Let $p-1$ of these subarrays consist of $\lceil n/p \rceil$ elements and the remaining subarray contains $n+1-(p-1)\lceil n/p \rceil (\leq \lceil n/p \rceil)$ elements.

Assign processor P_i to the subarray $A[i\lceil n/p\rceil, \ldots, i\lceil n/p\rceil + \lceil n/p\rceil - 1]$, for $i = 0, \ldots, p-2$, and assign P_{p-1} to the subarray $A[(p-1)\lceil n/p\rceil, \ldots, n]$.

2.- *Sort each subarray.* In parallel, each processor P_i, for $i = 0, \ldots, p-1$, sorts its subarray using a sequential *Runs*-optimal sorting algorithm.

3.- *Compute Runs*(A). In parallel, processor P_i, for $i = 0, \ldots, p-2$, checks if $A[i\lceil n/p\rceil + \lceil n/p\rceil - 1] > A[(i+1)\lceil n/p\rceil]$ and the "prefix sum" of these boolean values is computed. If A is sorted, terminate.

4.- *Merge the runs found in the previous step.* Use *Adaptive Parallel Merge* to merge the runs resulting from steps 1 to 3.

The merit of the algorithm is expressed in the following theorem.

Theorem 3.1 *Let X be a sequence of n elements from a total order.*

- *In a CRCW-P-RAM with $p = n^{1-\epsilon}$ processors $(0 < \epsilon < 1)$ Parallel Block Sort sorts X in $O((n/p)[1 + \log(Runs(X) + 1)] + \log p)$ time. The cost of Parallel Block Sort is $O(n[1 + \log(Runs(X) + 1)] + p \log p)$ units which is Runs-optimal if $Runs = \Omega([p/n] \log p)$.*
- *In a EREW-P-RAM or CREW-P-RAM with $p = n^{1-\epsilon}$ processors $(0 < \epsilon < 1)$ Parallel Block Sort sorts X in $O((n/p)[1 + \log(Runs(X) + 1)] + \log p)$ time. The cost of Parallel Block Sort is $O(n[1 + \log(Runs(X) + 1)] + p \log p)$ units which is Runs-optimal.*

Proof: Let X be a sequence of length n. The correctness of the algorithm follows from the fact that it stops after step 3 when A is sorted or it merges the resulting runs in step 4.

We now analyze the time used by the algorithm. Clearly, step 1 takes constant time on a *CRCW-P-RAM*, while it requires $\log p$ time on a *EREW-P-RAM* or on a *CRCW-P-RAM*. Since a sequential *Runs*-optimal algorithm is used in step 2 and each block $A[i\lceil n/p\rceil, \ldots, i\lceil n/p\rceil + \lceil n/p\rceil - 1]$, for $i = 0, \ldots, p-2$, and $A[(p-1)\lceil n/p\rceil, \ldots, n]$ is a subsequence of the original array, the time used in this step is $O((n/p)[1 + \log[1 + Runs(X)]])$. Now, let X' be the sequence in array A after step 2. A pair of contiguous elements $A[i], A[i+1]$ that delimit a run because $A[i] > A[i+1]$ will be called a "break-point". It is not hard to see that, after step 2, a block $A[i\lceil n/p\rceil, \ldots, i\lceil n/p\rceil + \lceil n/p\rceil - 1]$ has no break-point. All break-points in a block are removed, except for the first and the last which may be translated to positions between blocks. This implies that $Runs(X') \leq 2Runs(X)$. In step 3, the algorithm takes $\log p$ on a *CRCW-P-RAM* and $\log p$ on a *EREW-P-RAM* or on a *CREW-P-RAM*. If the algorithm stops at step 3, the theorem follows. In step 4, *Adaptive Parallel Merge* works with $Runs(X')$ sequences, thus it takes $O((n/p) \log[1 + Runs(X')]) =$

$O((n/p)\log[1 + Runs(X)])$ time. From this analysis and the fact that s sequential *Runs*-optimal algorithm sorts in $\Theta(n(1 + \log[1 + Runs(X)]))$ time [14], the theorem follows. □

4 Final remarks

We have presented adaptive parallel algorithms for optimal sorting with respect to *Dis*, *Max* and *Runs*. Our algorithms are designed for all types of *P-RAMs* with a fixed number of processors $p = n^{1-\epsilon}$ $(0 < \epsilon < 1)$ and can easily be adapted to handle the case $p = n$ (where n is the size of the input). This paper leaves three main lines for further research. First, to our knowledge no adaptive parallel algorithms for optimal sorting with respect to the measure *Inv* (the total number of inversions) and designed for a *P-RAM* with a fixed number p of processors has been presented yet. Second, it would be interesting to design adaptive parallel algorithms for optimal sorting with respect to several measures of disorder. Finally, we have discussed adaptivity when we have a sublinear number of processors. However, the following problem suggested by Akl [1] remains open: are there pallelel algorithm for sorting which, for some $0 \leq u \leq 1$, use $O(n/\log^{u-1} n)$ processors and run in $O(\log^u n)$ time?. We would be interested to know if such algorithms could profit from existing order in the input. If in this case the cost of discovering that the input is sorted is the same as sorting, then there is little hope for adaptive algorithms.

References

[1] S. G. Akl. Optimal parallel algorithms for computing convex hulls and for sorting. *Computing*, 33(1):1–11, 1984.

[2] S. G. Akl. *Parallel Sorting Algorithms.* Notes and Reports in Computer Science and Applied Mathematics. Academic Press, Orlando, Florida, 1985.

[3] S. G. Akl. *The Design and Analysis of Parallel Algorithms.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1989.

[4] S. G. Akl and N. Santoro. Optimal parallel merging and sorting without memory conflicts. *IEEE Transactions on Computers*, C-36(11):1367–1369, November 1987.

[5] T. Altman and B.S. Chlebus. Sorting roughly sorted sequences in parallel. *Information Processing Letters*, 33:297–300, 1989/90.

[6] T. Altman and Y. Igarashi. Roughly sorting: Sequential and parallel approach. *J. of Information Processing*, 12:154–158, 1989.

[7] G. Bilardi and A. Nicolau. Adaptive biotonic sorting: an optimal parallel algorithm for shared-memory machines. *SIAM Journal on Computing*, 18(2):216–228, 1989.

[8] S. Carlsson and J. Chen. An optimal parallel adaptive sorting algorithm. *Information Processing Letters*, 39:195–200, 1991.

[9] V. Estivill-Castro. *Sorting and Measures of Disorder.* PhD thesis, University of Waterloo, 1991. Available as Department of Computer Science Research Report CS-91-07.

[10] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. Technical Report CS-91-64, Department of Computer Science, University of Waterloo, 1991.

[11] A. Gibbons and W. Rytter. *Efficient Parallel Algorithms.* Cambridge University Press, Cambridge, 1988.

[12] C. Levcopoulos and O. Petersson. An optimal parallel algorithm for sorting presorted files. In *In proceedings of the 8th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 154–160. Springer-Verlag Lecture Notes in Computer Science 338, 1988.

[13] C. Levcopoulos and O. Petersson. A note on adaptive parallel sorting. *Information Processing Letters*, 33:187–191, 1989.

[14] H. Mannila. Measures of presortedness and optimal sorting algorithms. *IEEE Transactions on Computers*, C-34:318–325, 1985.

[15] R. McGlinn. A parallel version of Cook and Kim's algorithm for presorted lists. *Software — Practice and Experience*, 19(10):917–930, October 1989.

[16] K. Mehlhorn. *Data Structures and Algorithms, Vol 1: Sorting and Searching.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin/Heidelberg, 1984.

[17] O. Petersson. *Adaptive Sorting.* PhD thesis, Lund University, Department of Computer Science, 1990.

[18] C.C. Ribeiro. Parallel computer models and combinatorial algorithms. *Annals of Discrete Mathematics*, 31:325–364, 1987.

TEAM ALGORITHMS
Uma Aplicação para Fluxo de Potência Elétrica

Djalma M. Falcão [1] *e Benjamín Barán* [2]
Universidade Federal do Rio de Janeiro
Caixa Postal 68516 - CEP 21945
Rio de Janeiro - RJ - Brasil

Resumo

O trabalho apresenta os *Algorítmos Combinados*, mais conhecidos na literatura como *Team Algorithms* [12] estudando sua potencialidade no contexto da computação paralela. São apresentados exemplos demostrando que o *Team Algorithm* consegue resolver problemas que os algorítmos que estão sendo combinados não conseguem resolver individualmente.

Um *Team Algorithm* é utilizado para resolver o problema de *Fluxo de Potência Elétrica* combinando 2 algorítmos com características bem diferenciadas: o Desacoplado Rápido, derivado do Newton-Raphson, e uma variante do Método da matriz Y, usando o algorítmo de Jacobi.

Diferentes políticas na escolha dos *pesos* do algorítmo são discutidas e resultados experimentais são apresentados para 9 problemas tipos (Sistemas IEEE).

Palavras Chaves:
Team algorithms, paralelismo, fluxo de potência, algorítmos de Jacobi e Desacoplado Rápido.

1 Introdução

A primeira proposta para utilizar *Algorítmos Combinados* na solução do Fluxo de Potência aparece em 1971 [5] tentando combinar as excelentes características de convergência do algorítmo de Newton-Raphson, com a menor complexidade do Método de Jacobi. Os resultados apresentados no trabalho não eram desalentadores, mas na prática não chegaram a ter muita repercusão devido à tecnologia computacional da época, puramente seqüêncial, para a qual resultava um desperdício de memória ter varios algorítmos trabalhando concorrentemente.

Com a aparição dos computadores paralelos na década de '80, muitas idéias são revisadas. Em 1983 aparece um trabalho que postula novamente a utilização de algorítmos

[1] Prof. Adjunto da Univesidade Federal do Rio de Janeiro - Brasil.
[2] Prof. Adjunto da "Univesidad Nacional de Asunción - Paraguay".

combinados [12]; sendo que desta vez, para computadores paralelos de forma que cada processador trabalhe com um algoritmo só, eliminando assim o problema de desperdício de memória da proposta seqüêncial. Esse trabalho denomina a esta combinação de algoritmos como *Team Algorithm*. Passada quase uma década, a tecnologia da computação paralela está suficientemente amadurecida para tirar proveito da utilização dos *Team Algorithms* [8, 9]. O objetivo do presente trabalho é estudar a potencialidade das idéias de [12] e sua utilização na solução de problemas de Fluxo de Potência Elétrica.

O trabalho é dividido em 5 seções. A seção 2 trata das idéias gerais do *Algoritmo Combinado*. A seção 3 apresenta o problema do Fluxo de Potência Elétrica, deixando as alternativas mais complexas de solução para a seção 4. Finalmente, a seção 5 apresenta as conclusões do trabalho.

2 Team Algorithm. O problema.

Muitos dos problemas computacionais conhecidos podem ser resolvidos por métodos bem diversos, cada qual com suas vantagens próprias. A escolha do melhor método nem sempre é simples, pois qualquer que seja a escolha, algum outro método pode ter outras características desejadas. Desta forma, a idéia de combinar algoritmos para tentar obter as melhores características de cada método deu origem aos *Team Algorithms*. O primeiro trabalho neste sentido [12] propõe resolver

$$\mathbf{F(X)} \;=\; \vec{\mathbf{0}} \tag{1}$$

onde: $X \epsilon \mathbf{R}^n$, e $F : \mathbf{R}^n \longmapsto \mathbf{R}^n$, combinando as excelentes qualidades de convergência do algoritmo de Newton-Raphson com uma variante do Método do Gradiente.

A idéia é combinar os algoritmos usando *pesos* $\alpha(k)$ e $\beta(k)$ conforme à equação:

$$X(k+1) \;=\; X(k) \;+\; \alpha(k)\, \Delta X_{NR}(k) \;+\; \beta(k)\, \Delta X_G(k) \tag{2}$$

onde: $k = 0, 1, 2, \cdots$ representa o número da iteração e os $\Delta X(k)$ são as variações no vetor X calculadas por cada algoritmo na iteração k.

Em particular, para a escolha dos algoritmos acima mencionados:

$$\Delta X_{NR}(k) \;=\; -[J(k)]^{-1}\, F(\,X(k)\,) \tag{3}$$

$$\Delta X_G(k) \;=\; -\gamma\, [J(k)]^T\, F(\,X(k)\,) \tag{4}$$

onde: $J(k) = \left\{\frac{\partial F_i}{\partial X_j}\right\}$ é o Jacobiano calculado na iteração k e γ é um parâmetro de relaxação ($0 < \gamma < 1$) escolhido suficientemente pequeno de forma a assegurar a convergência.

A versão síncrona do algoritmo 2 é bem simples: um processo *Administrador* envia o valor da iteração em curso $X(k)$ a cada algoritmo para que estes calculem os ΔX em paralelo e combina os resultados escolhendo os *pesos* adequados. O algoritmo continua

iterando até que o resíduo $E(X) := ||F(X)||$ seja menor que um dado Erro Aceitável ε. A título de ilustração, seja

$$F(X) = \begin{bmatrix} F_1(X) \\ F_2(X) \\ F_3(X) \\ F_4(X) \end{bmatrix} = \begin{bmatrix} x_1 + 10x_2 \\ \sqrt{5}(x_3 - x_4) \\ (x_2 - 2x_3)^2 \\ \sqrt{10}(x_1 - x_4)^2 \end{bmatrix}, \quad X \in \mathbb{R}^4 \tag{5}$$

Inicialmente analisemos uma escolha a priori dos *pesos* sem variação ao longo da execução. Duas escolhas dos *pesos* são especialmente interessantes.

($\alpha = 1$; $\beta = 0$): que resulta no primer algorítmo, o Newton-Raphson,

($\alpha = 0$; $\beta = 1$): que resulta no algorítmo do Gradiente.

Consideremos os resultados experimentais de um caso típico usando o resíduo $||F(X)||_2$ e $\varepsilon = 0.01$.

Exemplo 1: $X(0) = [1\ 0\ 0\ 1]^T$
Para esta condição inicial, o algorítmo de Newton-Raphson não pode ser utilizado, pois o Jacobiano é singular. Por sua parte, o algorítmo do Gradiente não consegue convergir em menos de 200 iterações[3]. A alternativa de utilizar um *Team Algorithm* que permita sair da região onde o Jacabiano é singular e então aproveitar as excelentes características do Newton-Raphson, torna-se evidente. De fato, para $\alpha = \beta = 0.8$ o *Algorítmo Combinado* converge em 3 iterações.

O exemplo mostra claramente as vantagens do *Team Algorithm*. Para esclarecer o comportamento do algorítmo, pesquisamos o número I de iterações necessárias até a convergência, para diferentes *pesos*. Os valores da função:

$$I = f(\alpha; \beta) \quad > 0 \tag{6}$$

podem ser apresentados de forma gráfica no '*plano* $\alpha - \beta$', como ilustra o *mapa* da Figura 1.

Na Figura 1 pode-se notar uma *Região de Convergência Ótima* na qual se tem a melhor convergência possivel ($I = 3$). A medida que o ponto de trabalho se afasta desta região, o número de iterações I cresce.

Variando a condição inicial é posivel obter outros mapas similares [1]. Uma característica encontrada repetidas vezes em diversos problemas é que a *Região de Convergência Ótima* inclui alguns pontos da reta $\beta = 0$ que em geral ocorrem para $\alpha > 1$. Isto sugere que o algorítmo de Newton-Raphson pode ser acelerado usando um *Parâmetro de Sobre-Relaxação*[4] [2].Como exemplo, isto sucede com $X(0) = [3\ -1\ 0\ 1]^T$.

[3]O algorítmo do Gradiente requer 266 iterações.

[4]Idéia similar à utilizada nos algorítmos SOR (Successive Over-Relaxation).

β								
1.2	221	11			*	*		
1.0	266	10	6		*	*		
0.8	333	11	6	4	3	3		
0.6		11	6	4	3	5	33	
0.4			7	4	3	4	6	
0.2				5	3	3	4	
0	*	*	*	*	*	*	*	
	0	0.2	0.4	0.6	0.8	1.0	1.2	α

Figura 1: Mapa do Exemplo 1 ilustrando como varia o número de iterações I no plano definido pelos pesos α e β. O * indica que o algorítmo não converge para esse ponto.

Outros detalhes sobre as características dos *mapas* são discutidos em [1] onde são apresentados exemplos para os quais o número de iterações I é mínimo para valores negativos de β e $\alpha > 1$. Em outras palavras, a combinação ótima utiliza o Gradiente para diverger ligeiramente, de forma a permitir uma melhor convergência do Newton-Raphson. Uma situação que nos faz lembrar idéias do 'Simulated Annealing' [4].

Pode-se pesquisar o que acontece se o *Administrador* tem 'inteligencia' suficiente para escolher os melhores pesos a cada iteração, segundo um dado critério. Desta forma, os *Pesos* $\alpha(k)$ e $\beta(k)$ da equação 2 são função da iteração k. Os métodos para a escolha 'inteligente' dos *pesos* podem ser vários. Para o exemplo 1 foram testadas varias minimizações do resíduo E, em diferentes normas. A que apresentou o melhor desmpenho em todos os casos foi a que minimiza $||F(X)||_\infty$. Assim, no exemplo 1 se obteve o menor resíduo depois de 3 iterações.

Postulando um *Administrador inteligente*, utilizamos uma definição de erro E_r que utiliza a solução X_0 na etapa de *aquisição de dados* com o objetivo de descobrir a melhor escolha dos *Pesos* de forma a minimizar um *Erro Linear Er* definido como:

$$Er = \|X(k+1) - X_0\|_p = \|X(k) + \alpha(k)\ \Delta X_{NR}(k) + \beta(k)\ X_G(k) - X_0\|_p \qquad (7)$$

A minimização utilizando a equação 7 para $p = 2$ é conhecida na literatura [11] e se limita ao cálculo da *Pseudoinversa* $A^+ := (A^T A)^{-1} A^T$ da matriz A dada por:

$$A = \begin{bmatrix} \sum_{i=1}^n \Delta X_{NR}^2 & \sum_{i=1}^n \Delta X_{NR}\Delta X_G \\ \sum_{i=1}^n \Delta X_{NR}\Delta X_G & \sum_{i=1}^n \Delta X_G^2 \end{bmatrix} \qquad (8)$$

O método consegue excelentes resultados experimentais com uma baixa carga computacional. Assim, no Exemplo 1, converge em 3 iterações com o menor de todos os resíduos.

3 Fluxo de Potência em Redes Elétricas

O problema do Fluxo de Potência em Redes Elétricas consiste essencialmente na determinação do estado da rede (tensões em cada barra). O problema já foi amplamente tratado

na literatura [6, 10]. Do ponto de vista computacional, o problema consiste em calcular as tensões E conforme à equação:

$$YE = I(E), \quad \text{onde } Y \in \mathbb{C}^{N \times N}, \text{ e } E, I \in \mathbb{C}^N \tag{9}$$

I representa a corrente de cada barra, enquanto Y é conhecida como Matriz Admitancia. Esta matriz, além de ser simétrica, é altamente esparsa [7].

A tensão E_m da barra m, por ser complexa, pode ser expressa em coordenadas cartesianas e polares como:

$$E_m = Vr_m + \jmath Vi_m = V_m \exp(\jmath \Theta_m) \tag{10}$$

Para a solução do Problema do Fluxo de Potência existem diversos métodos [6, 10]. Dentre todos eles o mais utilizado é o Desacoplado Rápido [6], uma sofisticação do Newton-Raphson. Uma das características mais interessantes deste método é a de considerar de forma desacoplada os valores de V e Θ. O outro método a ser combinado será o Jacobi [5], uma versão do Método da Matriz Y apresentada em [10]. O *Algorítmo Combinado* resultante, à semelhança da equação 2, está dado por:

$$E(k+1) = E(k) + \alpha(k)\, \Delta E_{DR}(k) + \beta(k)\, \Delta E_J(k) \tag{11}$$

ou de forma equivalente:

$$E(k+1) = c(k)\, E(k) + \alpha(k)\, E_{DR}(k) + \beta(k)\, E_J(k) \tag{12}$$

onde: $c(k) = 1 - \alpha(k) - \beta(k)$.

Fazemos notar que a tensão $E_{DR}(k)$ calculada pelo Método Desacoplado Rápido está naturalmente em coordenadas polares, enquanto $E_J(k)$ calculado pelo Método de Jacobi, em coordenadas cartesianas. Por simplicidade, supomos nesta seção que os pesos são reais e constantes em cada iteração ($\alpha(k) = \alpha$ e $\beta(k) = \beta$). Assim, a equação complexa 12 pode ser re-escrita em suas componentes cartesianas:

$$\begin{cases} Vr(k+1) = cVr(k) + \alpha Vr_{DR}(k) + \beta Vr_J(k) \\ Vi(k+1) = cVi(k) + \alpha Vi_{DR}(k) + \beta Vi_J(k) \end{cases} \tag{13}$$

A equação 13 pode ser ligeramente modificada de forma a obter uma versão similar em coordenadas polares, dado que as barras PV são mais facilmente tratadas quando a tensão V_m aparece explicitamente.

$$\begin{cases} V(k+1) = cV(k) + \alpha V_{DR}(k) + \beta V_J(k) \\ \Theta(k+1) = c\Theta(k) + \alpha \Theta_{DR}(k) + \beta \Theta_J(k) \end{cases} \tag{14}$$

É importante salientar que as equações 13 e 14 não são em geral idênticas, mas estão muito próximas na medida que: $\Theta(k) \approx \Theta_{DR}(k) \approx \Theta_J(k)$

Os mapas obtidos utilizando coordenadas cartesianas e polares são apresentados em [1], onde pode-se notar que o número de iterações I é praticamente o mesmo para os dois casos encontrando-se algumas diferenças só no resíduo final. As mesmas características dos *mapas*, discutidas na seção 2, podem ser novamente verificadas para o problema do Fluxo de Potência.

4 Algorítmos Combinados de Pesos Variáveis em Problemas de Fluxo de Potência

Esta seção apresenta algumas metodologias de *pesos variáveis* usando minimização do resíduo. A equação 13 para coordenadas cartesianas, poderia ser re-escrita como:

$$\begin{cases} Vr(k+1) &= c_1(k)\,Vr(k) &+ \alpha_1(k)\,Vr_{DR}(k) &+ \beta_1(k)\,Vr_J(k) \\ Vi(k+1) &= c_2(k)\,Vi(k) &+ \alpha_2(k)\,Vi_{DR}(k) &+ \beta_2(k)\,Vi_J(k) \end{cases} \tag{15}$$

onde os pesos são agora função da iteração k e $c_i(k) = 1 - \alpha_i(k) - \beta_i(k)$, como já fora definido na equação 12.

A simplicidade do método da pseudoinversa quando usado com números reais, sugere a idéia de aplicar-lo separadamente para cada componente cartesiana da equação 15, mesmo que estas componentes não sejam independentes. O algorítmo assim resultante será chamado de XY.

Uma idéia que a priori parece melhor, é a utilização de coordenadas polares, pois já foi mencionado que para muitos problemas práticos as componentes polares são quasi independentes. Isto pode ser feito modificando ligeramente a equação 14, e utilizando separadamente a Pseudoinversa para cada componente. Este algorítmo será chamado de $V\Theta$:

$$\begin{cases} V(k+1) &= c_1(k)\,V(k) &+ \alpha_1(k)\,V_{DR}(k) &+ \beta_1(k)\,V_J(k) \\ \Theta(k+1) &= c_2(k)\,\Theta(k) &+ \alpha_2(k)\,\Theta_{DR}(k) &+ \beta_2(k)\,\Theta_J(k) \end{cases} \tag{16}$$

Para minimizar o Erro dado pela equação 7, para o caso complexo, uma vez mais temos que utilizar a Pseudoinversa A^+, que para o caso complexo está dado por $A^+ := (A^H A)^{-1} A^H$. Esta proposta será denotada como algorítmo A^+.

Uma última proposta analisada neste trabalho consiste em tentar minimizar de forma aceitável o resíduo E. Este algorítmo será denotado como S_∞.

Os resultados experimentais destas 4 propostas de *Algorítmos Combinados de Pesos Variáveis* se resumem na Figura 2, onde são comparados aos outros algorítmos de *Pesos Constantes*, como o Jacobi (J), o Desacoplado Rápido (DR), o melhor *Team Algorithm* denotado como (TEAM) e o resultado utilizando o Desacoplado Rápido Sobre-Relaxado (DR-SR). Os resultados da Figura 2 foram obtidos para os seguintes *Problemas Tipo*:
(1) IEEE 5-BUSBAR SYSTEM, sistema com 4 barras PV.
(2) Sistema de 5 barras modificado, com 1 barra PV.
(3) IEEE 6-BUSBAR SYSTEM, sistema sem barras PV.
(4) Sistema modificado de 6 barras.
(5) IEEE 14-BUSBAR SYSTEM, com uma barra PV.
(6) Sistema de 14 barras, de alta carga.
(7) Sistema crítico de 14 barras, altamente sobrecargado.
(8) Sistema de 30 barras, exemplo de Monticelli (PWRD-86).
(9) IEEE 57-BUSBAR SYSEM, com 6 barras PV.

Problema Tipo	*Team Algorithms*							
	Pesos Constantes				*Pesos Variáveis*			
	J	DR	DR-SR	TEAM	XY	$V\Theta$	A^+	S_∞
1	3(45)	3(96)	2(51)	2(06)	2(30)	2(31)	$\star$	2(23)
2	9(70)	3(27)	3(78)	2(88)	2(88)	2(88)	3(01)	2(58)
3	9(87)	8(89)	5(96)	5(96)	4(38)	6(80)	5(66)	7(58)
4	$\star$	20(80)	16(79)	11(99)	$\star$	20(97)	$\star$	18(99)
5	$\star$	4(11)	4(56)	3(82)	4(79)	4(52)	3(60)	3(43)
6	9(48)	24(83)	12(47)	8(56)	6(69)	5(61)	7(30)	7(61)
7	28(94)	35(95)	39(96)	24(89)	$\star$	$\star$	11(85)	15(62)
8	$\star$	3(63)	4(64)	3(63)	4(18)	3(90)	4(08)	4(05)
9	$\star$	4(30)	4(44)	4(30)	4(09)	4(08)	4(26)	4(83)

Figura 2: Comparação do número de iterações I requerido por cada algorítmo estudado para diferentes problemas tipo de Fluxo de Potência. Não-convergência em 100 iterações se denota com $\star$. O resíduo na solução, multiplicado por 10^5, é indicado entre parêntesis.

Na Figura 2 pode-se observar que nos primeiros 8 problemas tipo alguma versão do *Algorítmo Combinado* consegue superar qualquer um dos algorítmos sendo combinados, sendo o problema tipo No. (7) o mais significativo, pois o Jacobi converge em 28 iterações e o Desacoplado Rápido em 35, enquanto o *Team Algorithm* usando a Pseudoinversa Complexa consegue convergir em 11 iterações. O único caso onde nenhum ganho foi conseguido foi no problema tipo No. (8).

5 Conclusões

Os *Team Algorithms* apresentados na seção 2 resultaram promissores nos problemas que podemos chamar de 'difíceis', onde um ou ambos algorítmos da combinação falham em certas regiões por onde o algorítmo deve passar para chegar até a solução. Assim, o Exemplo 1 mostrou que é possível não só um menor número de iterações até a convergência, mais também ampliar o dominio de convergência.

Os resultados para o Fluxo de Potência resumidos na Figura 2 demostram a potencialidade dos *Team Algorithms*, permitindo obter uma melhor convergência em 8 dos 9 casos testados. No entanto, o dominio de convergência não foi ampliado. Isto porque a condição inicial para o Fluxo de Potência está relativamente perto da solução numa região bem comportada e o algorítmo Desacoplado Rápido converge sem problemas.

Para concluir, enfatizamos o crescente interesse nos *Team Algorithms*; como exemplo, é discutida em [8, 9] a posibilidade de combinar algorítmos tradicionais con novas técnicas de inteligencia artificial [4] como os *algorítmos genéticos*.

Referências

[1] Barán B. *Algorítmos Combinados. Uma aplicação para Fluxo de Potência Elétrica.* Relatório Técnico. Laboratório de Computação Paralela. COPPE. Universidade Federal de Rio de Janeiro, Brasil, 1991.

[2] Bertsekas, D. P. e Tsitsiklis, J. N. *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall Inc., 1989.

[3] Borges C.L. *Investigação do Desempenho de Métodos de Solução do Fluxo de Potência para Processamento Paralelo e Vetorial*, COPPE/UFRJ. Tese M.Sc. Engenharia Elétrica. Rio de Janeiro, 1991.

[4] Davis L. *Genetic Algorithms and Simulated Annealing*, Research Notes in Artificial Intelligence. Morgan Kaufmann Publishers Inc., Los Altos, California, 1987.

[5] Dusonchet Y. P., Taludkar S. N. e Sinnot H. E. *Load Flows using a combination of Point Jacobi and Newton's Methods*, IEEE Transactions on Power Apparatus and Systems, Vol. PAS-90, pág. 941-949. Julho, 1971.

[6] Monticelli A. *Fluxo de Carga em Redes de Energia Elétrica*, CEPEL. Editora Edgar Blücher Ltda. 1983.

[7] Morozowski M. *Matrizes Esparsas em Redes de Potência: Técnicas de Operação*, Livros Técnicos e Científicos Editora S.A. Eletrobrás. Rio de Janeiro, 1981.

[8] Ramesh V.C., Quadrel R., de Souza P. e Taludkar N. *Asynchronous Teams*, 1991 Summer National Meeting, American Institute of Chemical Engineers, Pittsburgh - U.S.A., 1991.

[9] Souza P.S. e Talukdar S.N. *Genetic Algorithms in Asynchronous Teams*, Proceedings of the fourth international conference on Genetic Algorithms. San Diego, California, 1991.

[10] Stott B. *Review of Load Flow Calculation Methods*, Proceeding of IEEE, Vol. 62, No. 7, Julho 1974.

[11] Strang G. *Linear Algebra and its Applications*, Segunda edição. Academic Press Inc. Florida, 1980.

[12] Taludkar S.N., Pyo S.S. e Mehrotra R. *Designing Algorithms and Assignments for Distributed Processing*, Electric Power Reserch Institute. Final Report. Preparado em a Universidade de Carnegie-Mellon, Pittsburg, Pennsylvania. Novembro, 1983.

COMMUNICATING APPLICATION MANAGEMENT : A SERVICE-ORIENTED SERVER MODELLING

Fabienne FAURE[*] [**], Thierry DESPRATS[*], Yves RAYNAUD[*]

[*] I.R.I.T./S.I.E.R.A. Université Paul Sabatier
118, Route de Narbonne, 31062 TOULOUSE Cedex, FRANCE
Tel : (33) 61 55 67 68
Fax : (33) 61 52 14 58
E-mail : faure@irit.fr

[**] HEWLETT-PACKARD FRANCE
Boulevard Steve Biko, 38090 Villefontaine, FRANCE
Tel : (33) 74 82 30 00

Abstract

This paper expresses the foundations of a property which aims at making entirely compatible various computers and softwares, and which is a condition to the achievement of any open distributed processing in the nextcoming years: **interoperability**. We define interoperability from three complementary levels: *communication support, transparency of exchanged data and accessibility to available services*. In relation to these three factors, we establish an overwiew of existing concepts and solutions that nowadays comply with interoperability requirements. The second part of the paper proposes a solution for monitoring of services used by end-users in a communication environment : a **service-oriented management system**. The described system aims at both providing end-users with information about accessible and available services and ensuring quality of provided services. We expose in this paper the basic architectural principles of our solution, and a general specification of the components.

Key words

Interoperability - Service - Service monitoring - Quality of communication system
Service-oriented management system

1. INTRODUCTION

In the context of heterogeneous network management and distributed processing, we deal with the problem of service management. More precisely, we consider end-user (application or user) requirements from a service viewpoint. These services may be typical OSI, Internet or DCE (OSF) services, or any other activity that satisfies a communication need. We are interested in management of configurations of used services, called profiles. Provide with end-users means to query accessible services and means to obtain the best initial profile of services, and subsequently ensure the dynamic monitoring of this profile, are the questions which guide our researches. This service-oriented approach aims at contributing to quality of provided services, for interoperability of communicating entities.

This paper expresses the foundations of a property which aims at making entirely compatible various computers and softwares, and which is a condition to the achievement of any open distributed processing in the nextcoming years: **interoperability**. Then, we focus on the problems of service accessibility and service monitoring which are required in a communication. We propose a ***service-oriented management system*** which solves these probems and thus contibutes to configuration and performance management functionalities relevant to open system management.

2. INTEROPERABILITY, CONDITION FOR OPEN SYSTEMS

By definition, an open system is one in which the components and their composition are specified in a non-proprietary environment, enabling this system to interoperate with any other peer system. The opening degree of a system can be viewed from three perspectives: **portability, integration and interoperability [1]**.

Portability refers to the ability of a system component to be used in various environments (i.e various vendors).

Integration concerns the consistency of the various human-system interfaces or APIs (Application Programming Interfaces). It allows a system component to be used in an uniform way whatever its geographical and hardware locations may be.

Interoperability is the ability of two peer entities involved in a communication to exchange information in a sensible way and to use, each of them with success, the functionalities of the other one [2]. Interoperability is based on the mutual understanding of cooperating entities. It aims at deleting the entity heterogeneity in order to supply with entities a common platform of comprehension and interaction.

Three necessary and complementary interoperability factors (Fig 1) are observed:

* the ***communication support*** (related to the transfer of informations),

* the ***transparency of exchanged data*** (related to representation and comprehension of exchanged data),
* the ***accessibility to available competencies*** of the communication environment (related to the necessary awareness of services for performing a communication).

This classification makes more accurate the previous definition of interoperability: *two end users (application processes or users) are interoperable if they exchange information in a sensible way and if each of them uses with success the functionalities of the other one.*

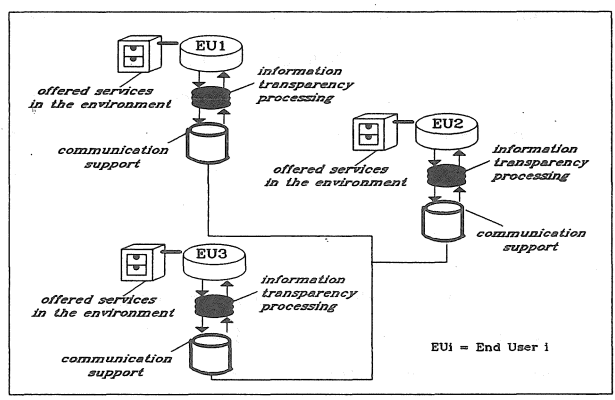


Fig 1: Interoperability factors

In relation to these three factors, we establish an overwiew of existing concepts and solutions that nowadays comply with interoperability requirements.

3. TAXONOMY OF INTEROPERABILITY

3.1. COMMUNICATION SUPPORT

The OSI Reference Model [ISO 7498-1], defines the seven-layered architecture from which are organized the open systems interconnection functions. The Physical, Data Link, Network and Transport layers focus on information communication, whereas the Session, Presentation et Application layers deal with information processing before this latter be really conveyed [3]. Nevertheless, the Session layer can be considered as the high part of the communication support since the virtual communication channel between two peer entities is managed in this layer. The Presentation and Application layers are more interested in data transparency and service accessibility factors.

Physical level

At the Physical level, interoperability seems well controled. The simple mention of the type of cable is often enough to define the entire physical appearance of the network

components and to specify what will connect and interoperate with what. This level of interoperability is made possible by efforts provided by organizations such as IEEE. The 802.3 (Ethernet-CSMA/CD), 802.4 (Token Bus) and 802.5 (Token Ring) LAN-intended standards specify exactly how should operate the physical layer of the network [4]

Data link level

In the Data Link layer, data is handled through formats which can differ according to underlying network architectures. Data link level interconnection solutions are **bridges**. This technology is used for extending a LAN to another LAN; it stores and forwards frames between networks. Most existing bridges require to operate on networks with similar or compatible architectures (eg: IEEE 802.x family) [5]. Networks with different architectures have to be interconnected at a higher level by other means, such as routers or gateways.

Network and Transport levels

At the Network and Transport levels, interoperability is based on semantic of bytes of a packet. In this context, the numerous Transport-layer definitions are not in favour of interoperability of systems. This plurality leads to a variety of record layouts and data fields meanings.

If a LAN has to be extended to a MAN or a WAN, the bridge technology is unsuitable. **Router** or even **gateway** technologies are necessary for interconnection of networks supported by different architectures. Routers [6] operate at the Network level. They identify the Network-layer protocols used by the two Network-entities connected, and ensure the routing for incoming packets

Every network architecture relies on particular protocols. These ones define the packet formats and the structures of handled addresses. The more usual are TCP/IP (Internet), XNS (Xerox), DECnet (DEC) and IPX/SPX (Novell). The existing router technology allows interconnection of network architectures unified around few de facto standards such as those mentioned above. In other case, specific gateways are required.

Session level

The Session layer main funtionality is to provide tools for dialogue management and synchronization [7]. Connection-oriented protocol and service have been specified for this layer by ISO [ISO 8326-8327]; a connectionless solution is currently studied [ISO DIS 9548-DAD3 8326].

The existing communication architectures show that the use of the Session layer is less obvious than the use of the further down layers. Numerous applications directly call for communication at the Transport-layer level. X-WINDOW, for example, is one of those

applications [8]. This is possible thanks to access interfaces such as Sockets or Transport Layer Interface [9].

The session concept is more concrete in the Systems Application Architecture (SAA) defined by IBM, thanks to its distributed transaction management solution: the Advanced Program to Program Communication (APPC) protocol specification and its implementation named LU6.2 [10]. Although this solution implements concepts specified in TP, the ISO application service element which deals with transaction processing [ISO 10026], it operates in an homogeneous environment built around LU6.2. Nowadays, the NetBIOS interface [11] is the best open solution corresponding to the OSI Session-layer. This API provides a set of services that allow a reliable full-duplex Session-level connection between two remote applications. NetBIOS is a de facto standard widely proposed upon LAN architectures.

3.2. TRANSPARENCY OF EXCHANGED INFORMATION

The second interoperability level concerns the exchanged message comprehension. Communicating entities must be able to receive or to send any data in a transparent and sensible way. An information format must be an obstacle neither for information communication nor for information interpreting. Therefore, the problem of various information representations that are met in an heterogeneous environment should be solved. This is the main function of the OSI Presentation-layer.

The OSI philosophy is the following: locally (i.e non-OSI world) the application processes use **concrete syntaxes** completely dependent on the hardware and software environment on which they run. Such syntaxes become unsuitable in an heterogeneous context, when remote processes have to communicate. Then, ISO proposes to hide the dependence between processes and their native environments. First of all, any couple of processes that wish to communicate should negociate one (or more) Presentation context. Such a context is defined by an **abstract syntax** (notation by which the process describes data units to be exchanged) and a **transfer syntax** (set of encoding rules which establish a common communication syntax for data units transmission) [7].

Both, a connection-oriented protocol [ISO 8822,8823] and a non oriented-connection protocol [ISO DP 9576] are specified for the Presentation level. In practice, de facto standard protocols that don't completely implement the OSI philosophy are proposed: XDR (SUN), Xerox Courier (Xerox) and NDR (Apollo/HP). They define coding rules for information representation (transfer syntaxes).

From an end-user viewpoint, the real benefits of interoperability result from the ability to deliver data, whatever their location may be, to anybody who need these data. Thus, the problem of information transparency joins the accessibility one.

3.3. ACCESSIBILITY TO OFFERED SERVICES

Interoperability is not only a sensible and synchronized information exchange on a reliable communication support. It includes capabilities thay any communication candidate entity should be aware of the offered and available services in the environment and the existing means to access them.

DCE [12], the OSF development platform for distributed environment, has been designed to provide both systems and applications interoperability. Among the DCE services, those relating to **communication** and **naming** offer solutions to problems of service accessibility.

OSF selected NCS 2.0 **RPCs** (DEC an HP) as the component for the DCE communication services [13]. RPCs aims at minimizing programming complexity relating to remote information transfer. This technology provides an identical behaviour regardless of the underlying transport protocol and ensures communication mechanisms transparency. RPCs mechanisms and ROSE [ISO 9072] have a similar scope: ROSE specifies an interaction support for remote communicating entities in conformance with the Client/Server model [ISO 10031-1], and RPCs are an implementation of this specification.

Naming services are particularly helpful in a distributed context. They allow objects (computers, files, etc) to be addressed regardless of their physical locations, by the use of user-oriented names which hide the access path complexity. Because of performance criteria, OSF has selected a local naming service (implemented on every cell of the distributed environment), and a global directory service for the whole environment. An application program uses the same interface to access either local or remote names. DIR-X (Siemens) is the remote naming service selected by OSF and is X500-compliant, the ISO directory service [ISO 9594]. For local names, OSF selected the cell naming service DECdns (DEC). The access interface to names is XDS (X/Open).

DCE provides accurate and concrete solutions which are helpul to solve problems of both applications and systems interoperability and supplies a global solution to distributed application programmers.

Our concern is to strengthen the accessibility aspect of interoperability with a ***service-oriented management system*** which manages end-user requests. This system aims at improving the use of offered services in the communication environment

4. SERVICE-ORIENTED MANAGEMENT SYSTEM MODELLING

Services are part of main actors that allow any organization to operate. Thus, their management is essential in order to allow an organization to operate in a consistent and suitable way. This management should ensure both **interoperability** and **quality of cooperation** within a network environment.

(Fig. 2) shows the proposed service-oriented system model. A ***service server*** allows end-users to be aware of all accessible services in the communication environment. Moreover, when several instances of a same service class are accessible in the environment, the server may select the instance the more convenient to an end-user requirement. It also ensures effectiveness of the offered service profiles.

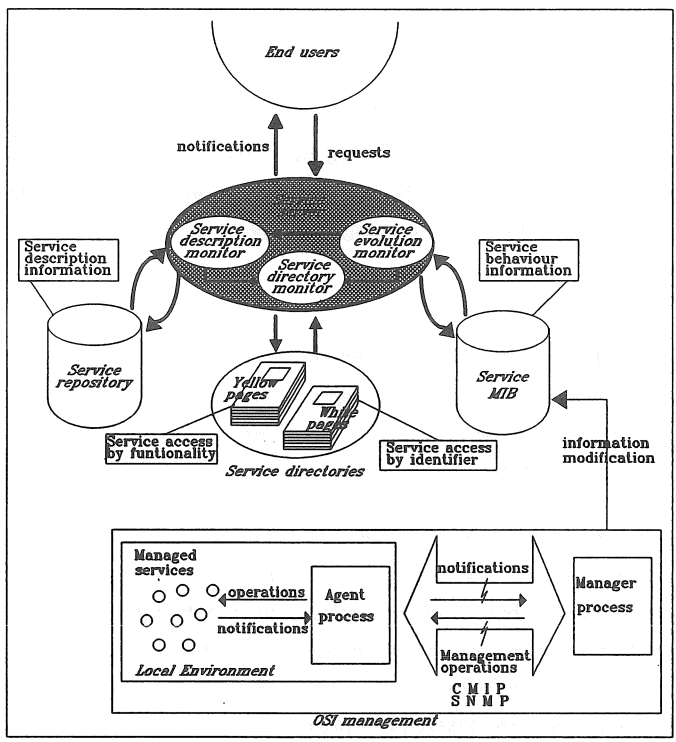


Fig 2: Service-oriented system modelling overview

4.1. WHAT ARE THE SYSTEM COMPONENTS ?

The system components may be classified within two groups: internal components and satellite components. ***Internal components*** define the management system kernel and perform all management functionalities. ***Satellite components*** supply useful information to internal components (Fig 2).

4.1.1. Satellite components

Satellite components of the service-oriented system are information storage tanks that allow internal components to operate. Three satellite components are identified:

The ***service repository*** contains all ***identification attributes*** (that describe a service in a static way) of monitored services,

The ***service MIB*** collects all ***status attributes*** (that describe a service in a dynamic way) related to service behaviour and performance.

Our concern can rely on the OSI management model, the basic principles of which are set in [ISO 10040], [ISO 10164-1..7] and [ISO 7498-4]. OSI management aims at providing mechanisms for interactive operating, monitoring and coordinating of resources. It defines solutions that ensure both management information access and transport [ISO 9595-9596]. The OSI management application relies on a management information base (MIB) [ISO 7498-4] which is a management object collection relevant to communication resources. Those objects are data supplied by the OSI management application components.

Service status attributes may be provided by a management application compliant with the OSI management model. Thus, beside the service-oriented management system, we consider an application which is responsible for both service activities observation and service MIB updating.

The ***service directories*** gather ***key attributes*** and allow to find a service location, either from its identifier (white pages principle), or from its functionality (yellow pages principles). This third component is useful to manage relationships between the service repository and the service MIB.

Convenient implementation solutions to these storage components are available. Databases may be used for identification and status attributes. More precisely, object-oriented and relational database management systems may be respectively used according to the complexity degree of structure object representation **[14]**.

X500 principles on which rely service directories may be achieved by using both DIR-X and DECdns included in the DME/DCE platform.

4.1.2. Internal components

Internal components define the service-oriented management system engine and are information customers. The highest level component is the ***service server*** we have mentioned abovel. It consists of the following sub_components:

The *service evolution monitor* consults and analyzes service management information (status attributes) collected in the service MIB in order to determine the required service availability, and eventually to select the most suitable service occurrence. Such a decision may be influenced by service location, service processing load rate, service failure statistics, and so on.

The *service description monitor* is responsible for management (consulting and updating) of service description information (identification attributes) gathered in the service repository.

The *service directory monitor* aims at refering and maintaining the service directories provided by naming service (yellow and white pages). More precisely, this monitor manages the service key attributes (service identifier/service functionality and service identifier/service location matchings). Functionalities of this monitor are used by both service evolution and service description monitors, when these latters have to search for a particular service.

4.2. WHAT ARE INTERACTIONS BETWEEN COMPONENTS ?

Interactions between internal components have to be completely defined. Satellite components rely on external tools to be used. An interaction between two components A and B is either considered as:

- a service request that A sends to B (if B is a processing component),
- an action that A performs on B (if B is a storage component),
- or a notification information that A sends to B.

Table 1 summarizes possible interactions which can be grouped in the three following classes:

4.2.1. Interactions between internal components themselves

The *service server* invokes services of the three monitors, according to end-user request semantics:

A consultation request aims at obtaining awareness of existing services in the environment. It requires *description monitor services*. These services may also be required for identification attributes updating.

An access request to a specified service requires *evolution monitor services* in order to both verify service availability and perform service selection if necessary.

Directory monitor services may be used in order to satisfy an updating request of key attributes stored in directories (identifiers, locations, functionalities).

Interaction issued from the service evolution monitor

The *service evolution monitor* uses *description monitor services* to retrieve complementary descriptive information about a service, in order to respond in a significant way to a service server request (eg: searching for a service selected authority).

Moreover, *directory monitor services* can be requested to provide a service real location address from this service identifier, or to obtain a service identifier from the service functionality.

The *service server* is notified with operation results obtained by the evolution monitor.

Interaction issued from the service description monitor

The *service description monitor* requests *directory monitor services* to quickly locate a service according to its functionality. It notifies the *service server* with obtained operation results or the *evolution monitor* with required service identification information.

Interaction issued from the service directory monitor

The *directory monitor services* sends service identification information to the *evolution monitor* or *description monitor*.

4.2.2. Interactions between internal components and satellite components

The *service evolution monitor* interacts with the *service MIB* by collecting service status attribute information to analyze required service availability and performance.

The *service description monitor* interacts with the *service repository* to provide service identification attribute information or update this latter when it is requested by the service server.

The *service directory monitor* interacts with the *service directories* to supply or modify a service identifier, a service location or a service functionality.

4.2.3. Interactions between internal components and end-users

End-users access the *service server* by service requests. Services server parses these requests and interact with end-users to notify them of a service result, a service failure or other notifications. The service server is the access point to the service-oriented management system.

INTERNAL COMPONENT	*INTERACTS WITH*
Service server	Service evolution monitor Service description monitor Service directory monitor End-users
Service evolution monitor	Service description monitor Service directory monitor Service MIB
Service description monitor	Service directory monitor Service repository
Service directory monitor	Service directories

Table 1: Internal component interactions

5. PERSPECTIVES: PERFORMANCE EVALUATION AND IMPLEMENTATION

The proposed service-oriented system aims at both providing end-users with information about accessible and available services and ensuring quality of provided services. We have exposed in this paper the basic architectural principles of our solution, and a general specification of the components.

We currently focus on a formal internal components modelling. Interface description tools such as IDL language [15] and simulation tools such as QNAP2 [16] help us to get a well defined component interfaces and to evaluate implementation performance. Thus, we want study the feasibility of the proposed model in terms of performance, particularly relating to response time and to the throughput introduced on the network.

Implementation perspectives lead us to consider a de facto standard open system, OSF/1 [17], which may support a development and management environment for distributed applications: OSF DCE/DME [12][18]. DCE offers solutions for both application and system interoperability. As soon as it is available we will implement processes that support the service-oriented system in this environment.

BIBLIOGRAPHY

[1] G.J. NUTT, **Open systems : integration, interoperability, portability,** Prentice Hall, 1991

[2] ISO/IEC JTC1/SC21 N4884, **Basic Reference Model of Open Distributed Processing,** July 1990

[3] A. TANENBAUM, **Computer networks,**Second edition, Prentice Hall, 1989

[4] P. ROLIN, **Réseaux locaux : normes et protocoles,** 3e Ed., HERMES, 1990

[5] M.A. MILLER, **Internetworking,** Prentice Hall 1991, A guide to network communications - LAN to LAN; LAN to WAN

[6] J.M. Mc QUILLAN, E.J. TEJA, Data Communication, September 29 1989, **Routers as building blocks for robust internetworks**
Router roundup : tools for network segmentation come of age

[7] H. NUSSBAUMER, **Teleinformatique III, Session, Presentation Compression de données, Couche d'Application,** Presses polytechniques et universitaires romandes, 1991

[8] T. DESPRATS, A. BENZEKRI, C. ARMENGAUD, **Ouverture des systèmes multifenêtre et intégration de technologies multimédia: situation et perspectives de normalisation,** Actes Convention Informatique Latine 91, Barcelone(Boixareu editores), Juin 1991

[9] W. R. STEVENS, **Unix network programming,** Prentice Hall, 1990

[10] J.P. DURAND, **Etude et réalisation de communications sous APPC/LU6.2 pour la mise en oeuvre de traitements coopératifs,** rapport interne SIERA, Juin 1991,

[11] B. GLASS, **Understanding NetBIOS,** Byte, January 1989, volume 14, number 1

[12] **Distributed Computing Environment Rationale,**Distributing Computing Environment Evaluation Team, Open System Foundation, 14 May 1990,

[13] B.HIRSCH, OSF DCE RPC Evaluation Report, Eureka Software Factory, 1991, **RPC Evaluation Criteria : a checklist of RPC**

[14] M. SIBILLA, F. FAURE, T. DESPRATS, A. VALDERRUTEN, **Information base modelling for open heterogeneous network management,** IFIP/IEEE Workshop on Distributed Systems : operation and management, DSOM'91, Santa Barbara, California, U.S.A., October 15-16, 1991

[15] **The Common Object Request Broker: Architecture and Specification**
OMG Document Number 91.12.1, Revision1.1, Draft 10 December 1991

[16] Bull, INRIA (1984) & SIMULOG (1990).

[17] **Guide to OSF/1 : a Technical Synopsis,** O'Reilly & Associates Inc, 1991

[18] M. AUTRATA, **OSF Distributed Management Environment,** IFIP/IEEE Workshop on Distributed System : operation and management, DSOM'91, Santa Barbara, California, U.S.A., October 15-16, 1991

[19] **HP OSF/1 Technology Release System,** Documentation Hewlett-Packard 1991, Programmer Guide, Programming Environment Reference, NCS 2.0 Programmer's Guide
NCS 2.0 Programmer's Reference

ABBREVIATIONS

DCE	Distributed Communication Environment
DME	Distributed Management Environment
IDL	Interface Definition Language
ISO	International Standard Organization
LAN	Local Area Network
MAN	Metropolitan Area Network
MIB	Management Information Base
NCS	Network Computing System
ODP	Open Distributed Processing
OMG	Object Management Group
OSF	Open System Foundation
OSI	Open System Interconnection
QNAP2	Queueing Network Analysis Package 2
ROSE	Remote Operatiion Service Element
RPC	Remote Procedure Call
SAA	Systems Application Architecture
TLI	Transport Layer Interface
WAN	Wide Area Netwok
XDS	X/Open Directory Service

Cache Memory for a LISP Machine

Feitosa, R. Q., Guidacci da Silveira, G. F.
Catholic University of Rio de Janeiro
Electrical Engineering Department
R. Marquês de São Vicente, 225
22 453 - Rio de Janeiro - RJ; Brazil
e-mail: RAUL@GSC.ELE.PUC-RIO.BR

Abstract

This work is part of a research to find out the appropriate cache strategies for LISP oriented archictectures. A virtual LISP machine, based on the SECD architecture, was developed. Trace driven simulation was the used methodology to evaluate different cache structures. This paper analyses how three aspects of the SECD architecture affect the cache performance: free list organization, the allocation of the cons cells and garbage collection. The simulation results show that linear free lists and a unique contiguous vector of cons cell enforce access locality. The garbage collection does not have a cumulative effect over the cache miss ratio. The analysis is based only on traces that do not contain intervening garbage collections. The cache miss ratio for a given test program varies within a limited range, as a result of successive garbage collections.

Keywords: LISP machine; garbage collection; cache; free list

1. Introduction

The Artificial Intelligence (AI) field has seen the development of new applications in the last 10 years. Among the AI languages, LISP is the most ancient, applied and known. But conventional Von Neumann machines, oriented to procedural languages like PASCAL and MODULA-2, are inefficient for LISP programs. Research efforts have been developed throughout the world to build appropriate machines for the fast execution of LISP programs.

At the Electrical Engineering Department of the Catholic University of Rio de Janeiro (DEE-PUC/RJ), several studies and prototypes have been developed around the SECD architecture. This machine, proposed by Landin [4] and extended by Henderson [3], was the basis for the first virtual machine implementation, the LispPUC

system [2]. After that, extensions for object oriented programming (LispObj) [9] and for logic programming (LispLog) [1] were developed. A concurrent LISP machine based on the SECD machine with shared memory and a concurent garbage collector was also object of study [5] [6]. Beyond that, a hardware microprogrammed implementation of the original SECD machine was developed with a parallel garbage collector [7] [8].

LISP programs present few logical and arithmetical operations and many memory accesses. The performance of LISP machines are much more dependent of the memory read and write response time than the conventional machines.

Cache memories are an economical and effective way to decrease the memory access mean time in conventional systems. Numerous studies were produced to determine the ideal structure for a cache memory. All these works were based on conventional machines and do not necessarily apply to LISP machines.

A research work to study structures and manegemment strategies of cache memories for LISP machines has begun at the DEE-PUC/RJ. This paper presents the first results of this research.

The SECD machine, implemented as a virtual machine in previous works at the DEE-PUC/RJ, was the basis for this study. The next section presents a résumé of the SECD machine, essential for the comprehension of the analysis that follows: the influence of the free list structure, the cons cell allocation strategies and the importance of the garbage collection on the performance of a conventional cache arquitecture.

2. The SECD Machine

The S.E.C.D. machine is defined basically by 4 (four) registers, a free list and the control part (see figure 1). The four registers **s**, **e**, **c**, **d** behave as pointers to lists and are named **Stack**, **Environment**, **Control List** and **Dump**. The **Stack** is used to hold intermediate and final results as the machine evaluates the expressions. The **Environment** holds the binding of variables during the computation of expressions. The **Control List** holds the program in the machine language during the execution. Finally the **Dump** saves the values of the four registers during a call to a new function.

The **ff** register points to the free list, known as the list of available cells. The machine state is completely defined by the contents of these four registers. Each

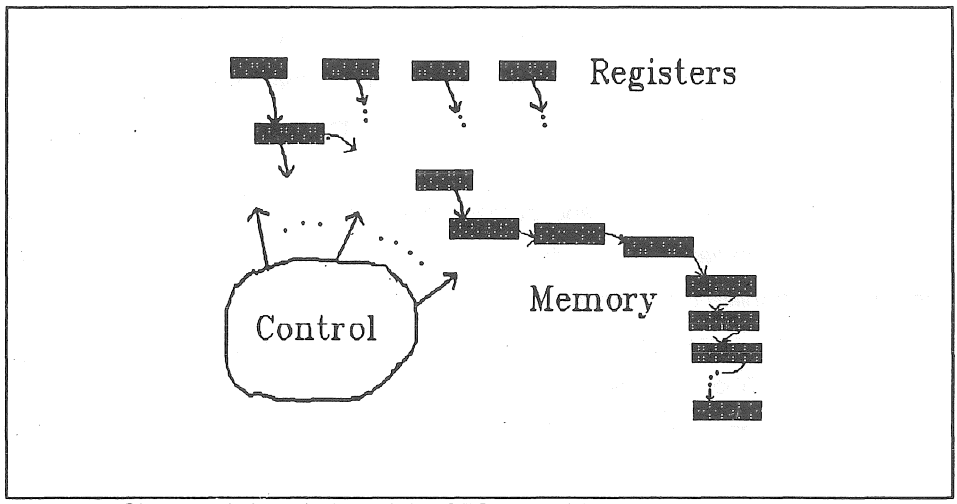


Figure 1: *Simplified description of the SECD machine.*

machine instruction is defined by the machine state before and after the execution of that instruction.

$$s\ e\ c\ d \xrightarrow{\text{instruction}} s'\ e'\ c'\ d'$$

The machine control (or the operational semantics) of this machine is defined by the set of all these state transitions. For each instruction, operator or basic LISP function defined in the LISP compiler, there is one (or more) correspondent instruction in the SECD machine.

The machine is organized as follows: the **Control List** points to the program in the machine code. The **Stack** points to the list of arguments. The other registers point to **nil**. The execution of a program in the SECD machine corresponds to a sequence of state transitions defined by the instructions in the **Control List**.

3. 0 Analyses of Access Locality in the SECD Machine

The garbage collection process is typical of LISP oriented systems. During the execution of a LISP program, lists are created and they remain in memory even after becoming useless for the active programs. The garbage collector recovers memory cells which contain the useless parts of the lists, making them available to new lists to be created. In sequential systems the program execution is suspended during the garbage collection. There are parallel LISP systems where program and garbage

collection can be executed concurrently. This work is limited to sequential systems.

The garbage collector produces a chained list - the free list - containing all free memory cells. On demand a cell is taken from the top of the free list. This procedure is repeated until the list runs out of free cells, causing the garbage collection to be re-executed.

Many cache structures and garbage collection policies were studied in the context of this research. Trace driven simulation was the performance analyses methodology applied throughout this work.

Four test programs were used in this study: a LISP Compiler, Tower of Hanoi, Queens and a Fibbonacci Series Generator. Traces were registered for different conditions. In all cases, they were taken during a period without intervening garbage collections. The traces have between 290,000 and 350,000 references.

3.1 The Free List Organization

The simulation results indicated that the cache performance heavily depends on the way the free list is organized in memory. To analyze this problem, two alternatives were considered:

linear free list: the nearer a cell is to the top of the list, the higher is the memory address where it is stored. In a dual organization, the free list starts at a lower address and extends to increasing addresses. It will be clear later that, for the purpose of this work, both organizations are equivalente. This is the most usual way to organize the free list in memory.

random free list: this organization is equivalent to spreading the list cells in a random way along the whole available memory. This organization is only considered with the aim to estimate the effect of the free list organization over the cache performance.

Both organizations were implemented in the virtual SECD machine and their corresponding traces were taken. Figure 2 shows the results from many simulations of a 8 Kbytes, two way set associative cache, for each trace. The curve for the linear list presents lower values than the curve for the random list. It is worth to note that the difference between these two curves becomes greater as the line size increases. This figure shows that the total number of misses can be two times greater for the random

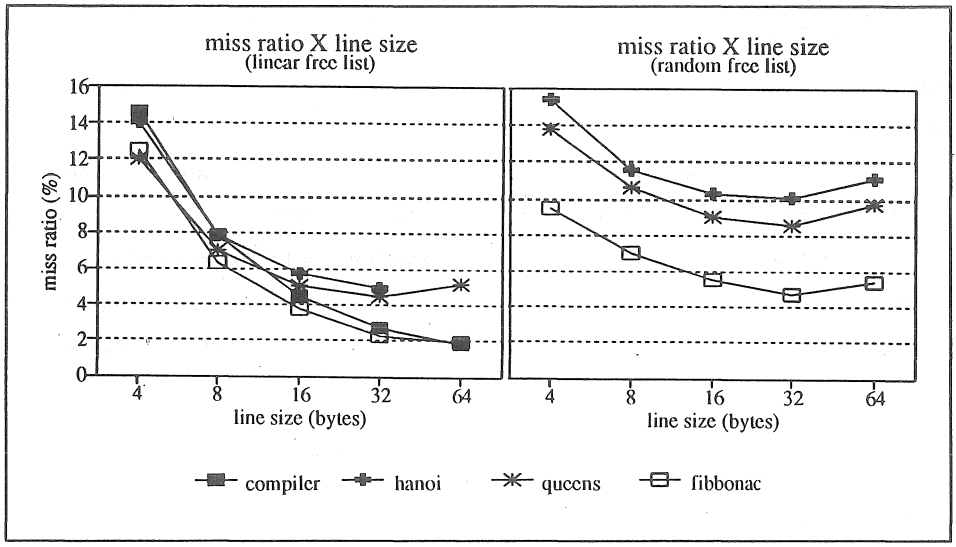


Figure 2 *Miss ratio as a function of line size, for a 8 Kbytes, two way set associative cache using write back, LRU for a a) linear b) random free list.*

free list. The significant miss ratio reduction obtained by doubling the line size clearly indicates that the linear free list enforces the spatial access locality.

These results have motivated a deeper analyses of the linear free list structure. An additional function was introduced in the virtual machine. This function makes a snapshot of the memory soon after the garbage collection is executed. The snapshots save the addresses of each free cell and its successor. It was verified that in more than 94% of the cases, consecutive list cells are stored in consecutive memory addresses. It was also observed that the number of cells stored in memory between two consecutive free list cells are in more than 99% of the cases not greater than 1. Due to this characteristic, the lists created during program execution tend to occupy a contiguous memory region. Therefore accesses to those lists are concentrated in a restricted memory area, in the sense of the spatial locality.

3.2 Allocation of the cons cells in memory

Relative to other cache organizations, direct mapped caches present higher miss ratios. However, their simplicity contributes to reducing the hit circle in the cache. Simplicity and the consequent speed make the direct mapping an interesting cache organization for real systems.

At a certain point of the research the simulations presented very high miss ratios (above 50%) for direct mapped caches. For the same conditions, a two-way set associative cache produced much lower miss ratios. This behavior is not observed in caches for conventional systems [13].

After many experiments changing the cache structure and the memory allocation strategy, the reason for these results became clear.

The following discussion is limited to the conventional list representation in LISP. Each cons cell consists of two fields, car and cdr, each one pointing respectively to the left and right list continuations. Alternative list structures for LISP are proposed in [10][11]. It is assumed that the cache address mapping function is bit selection [12].

Consider the following cons cells allocation. Two vectors, CAR[i] and CDR[i], are created in memory to store the car and cdr pointers. Each vector occupies a contiguous memory area starting respectively at the addresses *base_car* and *base_cdr*. Assuming that each vector element is one word long, the pointers CAR[i] and CDR[i] are stored respectively in the addresses *base_car* + *i* and *base_cdr* + *i*, for all i in the considered range.

Taking into account the discussion of the last section about linear free lists, one concludes that the content of CAR[i] and CDR[i] in most cases are equal to 1-α, where α is a natural number greater than zero and near to 1. This implies in a high probability that CAR[i] and CAR[CAR[i]] are stored in the same memory block. A similar statement applies also to CDR[i] and CDR[CDR[i]], to CDR[CAR[i]] and CDR[CDR[i]], and so on.

A simple inspection of LISP programs shows that walking through lists (list traversal) is a very frequent operation. Consider for example the path defined by the following function calls: car(cdr(car(i))). Assuming the described allocation, this operation can be executed just by accessing CAR[i], CDR[CAR[i]] and CAR[CDR[CAR[i]]], in this order.

According to the previous discussion, CAR[i] and CAR[CDR[CAR[i]]] probably belong to the same memory block. An unlucky choice for *base_car* and *base_cdr* may imply that the block containing CDR[CAR[i]] and the block containing CAR[i] and CAR[CDR[CAR[i]]] map into the same associative cache set. In a direct mapped cache, the second access (to CDR[CAR[i]]) will be a miss and the block containing CAR[i] will be displaced from the cache. The third access (to CAR[CDR[CAR[i]]]) will

be also a miss, and the cache will discharge the block containing CDR[CAR[i]] to make room for the new requested block. This can be seen as a competition between a block containing car-pointers and another block containing cdr-pointers for the same cache line. This causes a cache thrashing.

SET SIZE	TEST PROGRAMS											
	Compiler			Hanoi			Queens			Fibbonacci		
	WC	DT	CC	WC	DT	CC	WC	DT	CC	WC	DT	CC
1	49.5	6.1	4.1	49.7	9.7	6.5	47.7	10.9	6.4	53.4	4.3	3.9
2	5.9	2.8	2.3	11.3	7.4	4.4	10.7	8.5	4.4	4.4	3.1	2.7
4	3.5	1.7	1.6	8.5	5.8	3.8	8.7	7.7	3.0	3.3	2.8	2.5
8	1.6	1.6	1.6	6.4	5.2	2.5	8.1	7.4	2.6	3.0	2.8	2.5
16	1.6	1.6	1.6	5.8	4.4	2.2	7.9	6.9	2.2	2.9	2.8	2.5

Tabel 1: *Miss ratio for three different allocations of the cons cells for a 8 Kbytes, four way set associative cache with 32 bytes lines, write back and LRU.*

Table 1 shows, in the WC (worst case) columns, the miss ratios for this allocation strategy. Direct mapped caches have very poor miss ratios. On the other hand, if the set size is doubled, the miss ratio reduces dramatically for all the test programs. It should also be noted that any further doubling of the set size does not produce a such dramatic miss ratio reduction. It is easy to understand that the appointed cause for the cache thrashing does not exist for set sizes greater than 1.

If the addresses *base_car* and *base_cdr* map to sets which stay apart from each other in the cache, the behavior changes completely. The DT (distributed) columns of table 1 show the miss ratios for such allocation. Direct mapped caches have, for this allocation, a much better performance.

A third allocation strategy was considered. It consists of creating an unique vector of cons cells. The car and cdr pointers of each cons cell are stored in consecutive memory addresses. Clearly the competition between accesses to car an cdr does not exist in this case. The CC (car-cdr) columns of table 1 show the miss ratio for this allocation. For all test programs, cache performs better for the CC allocation. Specially for the test programs Queens and Tower of Hanoi, the miss ratio is much lower than for the other allocation strategies considered.

3.3 The Effect of Successive Garbage Collections over the Cache Performance

The lists created during program execution are temporary. Soon after the execution of a program finishes, all memory cells containing such lists are garbage to be recovered by the next collection. Garbage collection practically cancels the effects of the executed programs over the set of free memory cells.

Among the cells that are not collected, are those where the lists pointed by the four machine registes are stored. One could expect that these non collected cells would have a cumulative degradation effect over the miss ratios. The simulation results do not confirm such expectation.

The values presented in figure 1 correspond to traces taken soon after the machine has started its operation. The CC allocation strategy were used in this case. No more than 2 garbage collections has been executed before the traces were taken.

TEST PROGRAM	LINE SIZE (BYTES)									
	4		8		16		32		64	
	*	**	*	**	*	**	*	**	*	**
Compiler	14.6	14.7	7.8	8.0	4.4	4.6	2.7	2.8	1.8	1.9
Hanoi	14.0	14.0	7.9	8.0	5.7	5.6	4.9	4.7	5.4	5.3
Queens	12.1	12.8	7.1	7.4	5.0	5.3	4.4	4.7	5.2	5.3
Fibbonacci	12.5	10.8	6.4	5.6	3.8	2.9	2.3	1.6	2.0	1.2

Tabel 2: *Miss ratio for a 8 Kbytes, two way set associative cache with 32 bytes lines, write back, and LRU for *) few garbage collections **) after many garbage collections.*

To determine how much the garbage collection could affect the cache performance, traces of the same test programs were registered after 15 till 30 garbage collections. Table 2 shows the miss ratios obtained through simulation. Linear free lists were used in both cases. The difference in performance is little. For the Fibbonacci Series Generator the several garbage collections caused even lower miss ratios.

Although the garbage collection affects the access locality and the cache miss ratios, this effect is not cumulative and is limited with a small range of values.

4. Conclusions

This work has demonstrated that the structure of the free lists can have an important impact on the access locality in a LISP machine. Linear free lists enforce access locality and positively affect the cache miss ratios.

The performance of direct mapped caches strongly depends on the allocation of the cons cells in the memory. The option to build two separate vectors, one for the car- and another for the cdr-pointers, may cause cache thrashing. This problem can be avoided by an appropriate choice of the initial addresses of both vectors in memory. The simulations indicated that an allocation where the car- and cdr- pointers of the same cons cell are stored at consecutive addresses, performs better then the other allocations considered.

Finally, the simulations also showed that the successive garbage collections do not cause a cumulative degradation of cache performance.

Many other aspects related to the cache structure for LISP machine are still to be studied. Memory actualization, prefetch and replacement strategies, are some examples. In the context of parallel LISP machines there is a lot of other open questions.

Acknowledgements: The authors want to acknowledge the support of CNPq and Secretaria de Ciência e Tecnologia of the Brazilian Government.

5. Bibliography

[1] Corbucci, Dante. "LispLog: Uma Linguagem para a Programação Funcional e para a Programação em Lógica". M. Sc. Dissertation, Electrical Engineering Department, PUC-RJ, 1989.

[2] Guidacci da Silveira, G. F., "Uma Primitiva Não-Determinística para o LISP e sua Aplicação na Programação por Retrocesso," Simpósio Brasileiro de Inteligência Artificial (3º), Rio de Janeiro, 1986, pp. 236- 245.

[3] Henderson, P., "Functional Programming - Application and Implementation," Prentice-Hall International, 1980.

[4] Landin, P.J, "The Mechanical Evaluation of Expressions," Computer Journal, vol 6, no.4, 1963, pp. 308-320.

[5] Martins, W. S., "Um Sistema LISP Paralelo: Concepção e Simulação," M. Sc. Dissertation, Electrical Engineering Department, PUC/RJ, 1990.

[6] Martins, W. S. e Guidacci da Silveira, G. F., "Um sistema LISP Paralelo," Simpósio Brasileiro de Arquitetura de Computadores e Processamento Paralelo (III), Rio de Janeiro, 1990, pp. 238-251.

[7] Oliveira, F. S. G., "Compilador de Microcódigo para uma Máquina S.E.C.D.," Final Year Project, Departamento de Engenharia Elétrica, PUC/RJ, 1987.

[8] Oliveira, F. S. G., "ENTERPRISE, Uma Arquitetura LISP," M. Sc. Dissertation, Electrical Engineering Department, PUC/RJ, 1991.

[9] Silva, R. P., "LispObj: Uma Linguagem LISP com Extensões para a Orientação a Objetos," M. Sc. Dissertation, Electrical Engineering Department, PUC/RJ, 1989.

[10] Pleszkun, A. R., "An Architecture for Efficient Lisp List Access," Int. Symp. on Computer Architecture, 1986, pp.191-198.

[11] Potter, J. L., "Alternative Data Structures for List in Associative Devices," Proc. Int. Conf. Parallel processing, 1983, pp. 486-491.

[12] Smith, A. J., "Cache Memories," Computing Surveys, Sept. 1982, pp. 473-530.

[13] Smith, A. J., "Cache Evaluation and the Impact of Work Load Choice," Int. Symp. on Computer Architecture, 1985, pp. 64-73.

Revisión Mutua de Creencias: Aplicación a Bases de Conocimiento.

Lic. Eduardo Leopoldo Fermé
Lic. Osvaldo Omar González

G.I.D.I.A.
Grupo de Investigación y Desarrollo en Inteligencia Artificial

Dpto. Computación.
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Av. Güiraldes S/N (1428)
Ciudad Universitaria
Buenos Aires - Argentina

e-mail:
ferme@zorzal.edu.ar
osvaldo@dcfcen.edu.ar

Compendio

Se propone un modelo de Revisión de Creencias, llamado de Revisión Mutua, basado en el modelo AGM. Se proponen funciones de revisión para bases de conocimiento que cumplan los postulados RM y se muestran distintos resultados teóricos de interés.

Palabras claves:

IA y Cambio de Creencias, Base de Conocimiento, Modelo AGM, Atrincheramiento Epistémico.

I INTRODUCCION.

Veamos el siguiente ejemplo:[13]

- Juan le dice a Pedro: yo he nacido en Puerto Carreño.
- José le dice a Pedro: yo he nacido en Puerto Ayacucho.
- Marcelo le dice a Pedro: Juan y José son compatriotas.

Un día Pedro, consultando un Atlas ve, para su sorpresa que Puerto Carreño queda en Colombia y Puerto Ayacucho queda en Venezuela. Pedro entonces debe revisar sus creencias, ya que estas envuelven una inconsistencia. Esta historia nos muestra un diario ejemplo de "Revisión de Creencias", causada por una nueva información que contradice las creencias aceptadas hasta ese momento por una persona (o estado de creencias).

Las distintas teorías de Cambio de Creencia (entre ellas el modelo AGM) buscan modelizar estos cambios.

I.A El modelo AGM. [2, 3, 4, 5, 9, 14].

Representa un estado epistémico como un conjunto de creencias (c.c.) K, cerrado por consecuencia lógica. Presupone un lenguaje $\mathbb{L}$, en el cual se representan las sentencias de K. Reconoce tres cambios posibles La expansión (agregado de nueva información), la contracción (eliminar nueva información) y la revisión (información nueva que contradice la existente).

I.A.1 Postulados de Revisión.

Llamamos a una función de revisión * de $K \times \mathbb{L}$ en K, tal que K_A^* denota la revisión de K por A.

(K_1^*) K_A^* es un c.c.

(K_2^*) $A \in K_A^*$

(K_3^*) $K_A^* \subseteq K_A^+$

(K_4^*) Si $\neg A \notin K$, entonces $K_A^+ \subseteq K_A^*$

(K_5^*) $K_A^* = K_\perp$ sii $\vdash \neg A$

(K_6^*) Si $\vdash A \Leftrightarrow B$, entonces $K_A^* = K_B^*$

I.A.2 Relación entre Contracción y Revisión.

Sean "+" una función de expansión y "-" de contracción, ambas de $K \times \mathbb{L}$ en K, cumplen los postulados respectivos de expansión y contracción [2,3,16]. Es posible definir contracción en términos de revisión y viceversa , tal cual lo ilustran las siguientes

fórmulas de Levi y Harper, respectivamente:

$$(\text{Def } *) \quad K_A^* = (K_{\neg A}^-)_A^+ \qquad\qquad (\text{Def } -) \quad K_A^- = K \cap K_{\neg A}^*$$

II MOTIVACION DEL TRABAJO.

II.A Discrepancias con AGM.

La existencia del postulado K_2^* trae implícita una primacía de la nueva información por sobre la ya existente. Esto nos resulta antiintuitivo, ya que no es lo usual en entes racionales; quienes en la realidad tienden generalmente a hacer una evaluación del nuevo conocimiento y, eventualmente, podrían llegar a rechazarlo total o parcialmente. En estos casos AGM no es aplicable.

III REVISION MUTUA (RM).

III.A Postulados Básicos

Sea K un c.c., μ una sentencia del lenguaje $\mathbb{L}$. Si RM() es una función de $\mathbb{K} \times \mathbb{L}$ en $\mathbb{K}$, donde $\mathbb{K}$ es un conjunto de conjuntos de creencias , entonces RM(K,μ) denota la revisión mútua entre K y μ.

En el primer postulado requerimos:

RM(1) RM(K,μ) es un c.c.

Antes de continuar con la enunciación, definiremos:

Sea X una sentencia del lenguaje $\mathbb{L}$, entonces el conjunto VAR(X) es el que se forma con todas las variables proposicionales que aparecen en X. Definimos consecuencias debilitadas de X, Cd(X), de la siguiente manera:

$$Cd(X) = \{ A \in \mathbb{L} \ / \ A \in Cn(X) \text{ y } VAR(A) \subseteq VAR(X) \}$$

Exigiremos que cualquier porción del nuevo comocimiento, representada por una fórmula $A \in Cd(\mu)$, sea aceptada o rechazada en la Revisión Mutua.[1]

---

1 Este postulado es un claro debilitamiento de K_2^* de AGM. En RM el imput es también "revisado".

RM(2) Si $A \in Cd(\mu) \Rightarrow A \in RM(K,\mu)$ o $\neg A \in RM(K,\mu)$

Un corolario de este postulado es el siguiente:

$$\mu \in RM(K,\mu) \vee \neg\mu \in RM(K,\mu)$$

que nos muestra que dada una entrada epistémica μ, podemos aceptarla o rechazarla, pero no permanecer indiferentes.

Si μ no contradice las creencias de K (esto es $\neg\mu \notin K$), la revisión es equivalente a la expansión. Esto viene expresado en los siguientes postulados:

RM(3) $RM(K,\mu) \subseteq K^+_\mu$

RM(4) Si $\neg\mu \notin K \Rightarrow K^+_\mu \subseteq RM(K,\mu)$

El quinto postulado plantea en que casos la revisión mutua dará como resultado el conjunto inconsistente

RM(5) Si $RM(K,\mu) = \perp$ entonces o bien $\vdash \neg\mu$
o bien $K = K_\perp$

Es importante destacar que esta condición es necesaria, pero no suficiente; ya que puede ocurrir que $K = K_\perp$, $\vdash \neg \mu$ y, sin embargo $RM(K,\mu) \neq K_\perp$.

El sexto postulado nos habla de la independencia de la sintaxis. Formalmente:

RM(6) Si $\vdash \mu \Leftrightarrow \sigma$ y $VAR(\mu) = VAR(\sigma)$ entonces $RM(K,\mu) = RM(K,\sigma)$.

III.B Epistemic Entrenchment para RM. (EE-RM)

Basados en las ideas originales de EE para AGM [2, 5, 9], definiremos un orden epistémico para las sentencias del conjunto $K \cup Cd(\mu)$.

El primer postulado requiere que $\leq$ satisfaga el mínimo requerimiento de una relación de orden:

EE-RM 1 (Transitividad)

Para todo A, B, C si $A \leq B$ y $B \leq C$ entonces $A \leq C$

La segunda condición será la existencia de una sentencia cuyo atrincheramiento epistémico sea mayor que el del resto. Se verá la importancia de este postulado al construir funciones de RM usando EE-RM.

EE-RM 2 (Existencia de supremo)

$\exists B \in K \cup Cd(\mu)$ tq. $B > A \quad \forall A \in K \cup Cd(\mu)$

De este postulado se desprende la siguiente propiedad:

Sea B el elemento supremo descripto en EE-RM 2; luego:

Si $\vdash \neg B$, entonces $RM(K,\mu) = \bot$

El tercer postulado plantea lo siguiente:

EE-RM 3 (Dominancia)

Para todo A, B y A no es supremo, si $A \vdash B$ entonces $A \leq B$

Una propiedad interesante que se desprende es la independencia de las sintáxis de las sentencias, es decir el ordenamiento es por clases de equivalencia lógica.

Formalmente:

Si $\vdash A \leftrightarrow B$, entonces $A \leq B$ y $B \leq A$

EE-RM 4 (Conjuntividad)

Para todo A, B ambos no supremo y $B <> \neg A$ entonces $A \leq A \& B$ o $B \leq A \& B$

Este postulado nos dice que si deseamos remover durante el proceso de RM a A & B, alcanza con remover A o B y no necesariamente ambos. Es decir (a partir de EE-RM 3) que A & B tendrá el valor epistémico del menor (bajo $\leq$) entre A y B

El quinto postulado pide decidir en caso de conflicto:

EE-RM 5 (Decidibilidad)

Si $A \in K \cup Cd(\mu)$ y $\neg A \in K \cup Cd(\mu)$, entonces $A < \neg A$ ó $\neg A < A$

El último postulado nos dice que aquellas sentencias de valor epistémico más alto, a excepción del supremo serán las tautologías

EE-RM 6 (Supremacia de las tautologias)
Para todo A, B ambos no supremo entonces:

Si $B \leq A \ \forall \ B$ entonces $\vdash A$

Notar que la recíproca es cierta a partir de EE-RM 3.

III.C Construcción de funciones.

III.C.1 Función usando EE-RM.

Dado K conjunto de creencias y μ una sentencia del lenguaje $\mathbb{L}$, definimos el siguiente procedimiento de construcción:

Ordenar las proposiciones de $K \cup Cd(\mu)$, según un ordenamiento total $\leq$, tal que:

$$X_n \leq X_{n-1} \leq \ldots \leq X_1 < X_0$$

$RM_0(K,\mu) := Cn(X_0)$

$$RM_i(K,\mu) \begin{cases} Cn(RM_{i-1}(K,\mu) \cup \{ X_i \}) & \text{si } (RM_{i-1}(K,\mu) \cup \{ X_i \}) \neq \bot \\ Cn(RM_{i-1}(K,\mu)) & \text{sino.} \end{cases}$$

$RM(K,\mu) := RM_n(K,\mu)$.

Esta función tiene el siguiente inconveniente: Si $\neg \mu \in K$ y además $\mu \in RM(K,\mu)$, entonces la revisión obtenida resulta maximal.

Esto no ocurriría si K fuera cerrado bajo Cd en lugar de bajo Cn. Esto hace que dicha función pueda ser computada, ya que los lenguajes de programación por lo general restringen la noción de Cn a Cd.

III.C.2 Identidad de Levi para RM (GENERALIZACION DE LEVI).

En I.A.2 vimos un teorema de Representación planteado por la identidad de Levi. Dicha identidad garantizaba en AGM la relación entre las funciones de contracción, expansión y revisión. Para el caso de RM, creemos necesario establecer su correlato en lo que definimos como "Generalización de Levi", la que nos caracteriza la

función de Revisión Mutua en términos de la contracción y la expansión.

La Identidad de Levi para RM podríamos establecerla como sigue:

$$RML(K,\mu) = \begin{cases} K^{+}_{\mu} & \text{si } \neg\mu \notin K \\ K_{j} \quad \text{con } J = \text{máx } i & \text{si no} \end{cases} \text{ ; donde:}$$

$$K_0 = (K^{-}_{\neg\mu})^{+}_{X_0} \qquad \text{y} \qquad X_0 \begin{cases} \mu \text{ si } \neg\mu < \mu \\ \neg\mu \text{ si } \mu < \neg\mu \end{cases}$$

con '<' una función de orden definida para $K \cup Cn(\mu)$.

Para $i > 0$, X_i cumple:

1. $X_i \equiv X_n \quad n = 0,..,i-1$
2. $X_n \in Cn(\mu)$ si $\mu < \neg\mu$ ó
   $X_n \in Cn(\neg\mu)$ si $\neg\mu < \mu$
3. $\neg X_n \notin K_{n-1}$

y $K_i = (K_{i-1})^{+}_{x_i}$

III.D. Teoremas de Representación y relación con AGM.

TEOREMA III.D.1

⊛ Sea P el procedimiento descripto en III.C.1, entonces existe un ordenamiento ≤ que cumple EE-RM1..EE-RM6 tal que la función $f(K,\mu)$ que surge a partir de la aplicación del procedimiento III.C.1 vía ≤, cumple con los postulados RM(1)..RM(6).

TEOREMA III.D.2

⊛ Sea $f(K,\mu)$ una función de revisión que satisface RM(1)..RM(6), y el procedimiento descripto en III.C.1 , entonces existe un ordenamiento ≤ que cumple EE-RM1..EE-RM6 tal que f es representada por el procedimiento vía ese ordenamiento.

TEOREMA III.D.3

⊛ Sean '-' y '+' funciones de contracción y expansión. Entonces, si '-' cumple los postulados $K^{-}_{1}..K^{-}_{6}$ y '+' cumple $K^{+}_{1}..K^{+}_{6}$ de AGM,

la función RML(K,μ) planteada a partir de la identidad de Levi de III.C.2 cumple los postulados RM(1)..RM(6).

TEOREMA III.D.4

⊛ Toda función de revisión f(K,μ) que satisface K_1^* - K_6^* de AGM es una función de RM que satisface RM(1)-RM(6).

III.E. ALGORITMOS COMPUTACIONALES

Dado el carácter algorítmico de las funciones descriptas, resulta fácil construir a partir de ellas un algoritmo computacional. Algunos ejemplos pueden verse en [17].

IV CONCLUSIONES

Definimos un nuevo modelo de cambio de creencias, inspirado en el modelo AGM, pero haciendo hincapié en las críticas formuladas en II.A. El resultado es una teoría de cambio de creencias cuya función de revisión no privilegia el *nuevo conocimiento*, sino que este es "revisado" a la luz del *viejo conocimiento*. Un resultado interesante es que el modelo AGM pasa a ser un caso particular de revisión mutua.

V AGRADECIMIENTOS.

Agradecemos a R. Rodriguez y M. Coniglio, integrantes del grupo *GIDIA* por su aporte en la corrección de este trabajo. A L. Barnator por su colaboración, y especialmente a C. Alchourrón y a R. Carnota por sus valiosos aportes.

VI BIBLIOGRAFIA.

[1] Fagin, R.; et al. (1983) "On the semantics of updates in databases", Second ACM SIGACT-SIGMOD. Ney York. 352-365

[2] Gärdenfors, Peter (1988), "Knowledge in flux", Massachusetts Institute of Tecnology.

[3] Alchourrón, Carlos; Makinson, David; (1982) "On the logic of theory change: Contraction function ..." , Theoria 48: 14-37

[4] Alchourrón, Carlos; Makinson, David; (1980) "Hierarchies of regulations and their logic", New Studies in Deontic Logic,

R.Hilpinen, ed. Dordrecht: Reidel, 123-148.
[5] Alchourrón, Carlos; Makinson, David; (1985) "On the logic of theory change: Safe contraction", Studia Logica 44, 405-422
[6] Gärdenfors, Peter (1990) "The dynamics of belief system: Foundations vs. coherence theories.", Reveu Trimestrielle 172.
[7] Grove, Adams (1986). "Two modelling for Theory Change.", Auckland Philosophy Paper 13
[8] Nebel, Bernhard (....) "A Knowledge Level Analysis of Belief Revision". Technical University of Berlin.
[9] Gärdenfors, Peter; Makinson, David (1988) "Revision of knowledge system using epistemic entrechment", Proceding of the Second Conference on Theorical Aspect of Reasoning About Knowledge, Morgan Kaufmann Publ., Los Altos.
[10] Mendelzon, Alberto; Katsuno, Hirofumi (1990) "Propositional Knowledge Base Revision and Minimal Change". Un.of Toronto.
[11] Dalál, Mukesh (1989) "Investigation into a theorie of knowledge base revision: Preliminary Report." Rutgers University.
[12] Fuhrman, Morreau [eds.] "The logic of theory change". Workshop, Ronstan, FRG (1989). Recopilación.
[13] Fermé, Eduardo [1992] "Actualización de Bases de Conocimiento usando Teorías de Cambio de Creencia" III Ibereamia '92. Cuba.
[14] Coniglio Marcelo, González Osvaldo [1991] "Revisión de Creencias". Curso GIDIA Universidad de Buenos Aires.
[15] Mendelzon Alberto, Katsuno Hirofumi [1990] "On the Difference Between Updating ..." University of Toronto.
[16] Kvitca, Adolfo.[1990] "Integrating Tableaux and Resolution Methods". Internal Report GIDIA
[17] González Osvaldo, Fermé Eduardo. "Construcción de algoritmos computables usando Revisión Mutua". Int.Report GIDIA.

Titulo : " UMBRALIZACION ADAPTATIVA EN TIEMPO REAL ".

Por :

M.A. Fernández (*) y J. Mira (**).

(*) Departamento de Informática. Universidad de Castilla la Mancha. E. U. P. A.
Campus Univ.- Avda. de España s/n. Albacete (02006).

(**) Departamento de Informática y Automática. Facultad de Ciencias Físicas de la U.N.E.D.
Aptdo. 60141. Madrid (28040).

Abstract:

Este trabajo presenta un algoritmo para el cálculo de umbral adaptativo en cada punto de imagen. Se realiza un estudio sobre la realización hardware para su funcionamiento en tiempo real, y se comentan los resultados obtenidos mediante una simulación de este sistema realizada en C sobre una tarjeta comercial de digitalización de imagen. Se comentan también otras posibles aplicaciones de este algoritmo en diferentes campos de tratamiento de la señal.

Palabras Clave: UMBRALIZACION / ADAPTATIVA / TIEMPO REAL / TRATAMIENTO DE SEÑAL / VISION / SUAVIZADO / PASO BAJA.

Planteamiento del problema:

Cuando intentamos clasificar los puntos de una imagen, umbralizando alguna de sus características, de nivel de gris o el valor obtenido al someter la imagen a cualquier tipo de filtro, de borde, o de otro tipo [1]; el cálculo del umbral, juega un papel decisivo en el éxito de nuestro propósito [2] [3]. En multitud de escenas, aparecen zonas de imagen de diferente naturaleza en las que el parámetro clasificador toma valores muy distintos. Por lo tanto,

si deseamos clasificar adecuadamente toda la imagen, el cálculo del umbral debe realizarse atendiendo al comportamiento del entorno del punto a umbralizar, ya que en la imagen completa aparecerán multitud de elementos que nos llevarían a definir un umbral " general ", poco fino para las zonas de imagen en las que el parámetro clasificador toma valores mas sutiles [4]. Un ejemplo claro de este caso, sería la detección de un misil o blanco lejano sobre un fondo celeste, acompañado en la escena por elementos de nubes, oleaje u horizontes montañosos.

El propósito de este trabajo es, producir un sistema capaz de resolver este problema trabajando en tiempo real. Supondremos inicialmente, que los datos que vamos a procesar proceden de una cámara de video con un estándar de señal CCIR. Esto nos dá una frecuencia de información, con una digitalización a 10 Mhz., de un nuevo dato cada 100 nseg, bien sea este dato un nivel de gris, bien corresponda a la respuesta de un filtro actuando sobre sobre la señal original.

En 100 nseg., no es fácil el análisis del histograma del entorno del punto [5], por lo que planteamos, para el ajuste de umbral, calcular la media de los valores del entorno del punto y definir el umbral ponderado este valor [6].

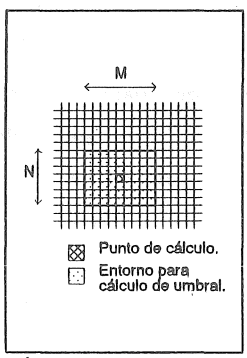


Fig. 1. Entorno.

Hemos considerado en este trabajo, como entorno que interviene en el cálculo de umbral para cada punto de imagen, el conjunto de puntos que le rodea, imponiendo para este entorno una geometría rectangular. Como podrá verse mas adelante, nuestro sistema es flexible en cuanto a las dimensiones del rectángulo que definimos como entorno, así como al lugar geométrico que ocupara el punto de cálculo dentro de este (ver fig. 1).

El cálculo que deberemos realizar, sera pues la suma de los valores de los puntos del entorno definido alrededor del punto de cálculo, dividir por el número de éstos, y ponderar el resultado mediante una constante para generar un umbral más o menos exigente. No es pues la dificultad del cálculo lo que complica la solución, sino el volumen de éste. Si tomamos como dimensiones para el entorno, por ejemplo 10 por 10 puntos de imagen, tendremos que operar con 100 sumandos, ponderar este valor y luego

comparar con el umbral en tan solo 100 nseg. Estas son dimensiones un tanto reducidas. Sería lo normal, dimensiones de entorno mayores como 40 por 30 o 70 por 100, por ejemplo, esto nos llevaría a tener que operar con 1200 o 7000 sumandos cada 100nseg., si deseamos realizar el cálculo en tiempo real.

Algoritmo para computación en tiempo real :

Podemos dividir el proceso en cuatro etapas distintas.

1) Realización del sumatorio de todos los puntos del entorno:

Proponemos para realizarlo, el siguiente algoritmo que denominamos de "Cómputo Acumulado".

Descompondremos el sumatorio como indica la figura 2.

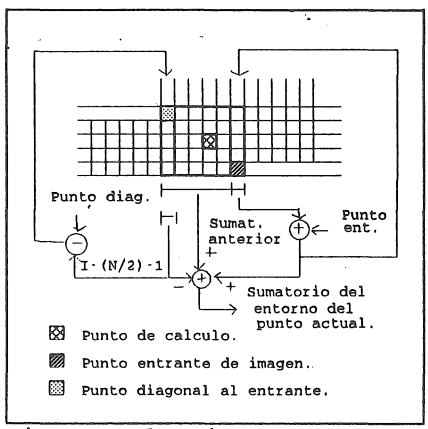


Fig. 2. Algoritmo.

El Cálculo se simplifica, si consideramos que al realizar el sumatorio de los valores del entorno de un punto de imagen, podemos utilizar los cálculos que realizamos para su vecino inmediatamente anterior. Es decir, para calcular el nuevo sumatorio, tan solo tendremos que restar al anterior la columna de entorno que desaparece de éste, y sumarle los valores de la columna que para el nuevo punto de cálculo entra a formar parte del entorno. Sería, por tanto, necesario almacenar los datos requeridos para realizar estas operaciones. Este almacenamiento se realiza mediante memorias tipo FIFO.

El grado de complicación del cálculo realizado, así como el volumen de este, no dependen de el tamaño del entorno que elijamos, ni de la posición que ocupe dentro de este el punto a umbralizar.

2) Cálculo del valor medio:

Podemos optar en este punto, por dividir el sumatorio realizado por el numero de puntos, o multiplicar por este el valor del punto de cálculo.

Para los puntos cercanos al borde de imagen, se genera un tratamiento especial que realiza la media, solamente con los puntos de su entorno que

forman parte de la escena.

3) Cálculo del umbral. El tipo de umbral a utilizar dependerá del problema concreto. Nosotros hemos trabajado con dos tipos de umbral:

- En un caso fijamos el umbral, incrementando la media calculada, con una constante fija en toda la imagen.

- En el otro caso incrementamos la media con una constante distinta en cada punto y proporcional a la media obtenida para su entorno.

4) El ultimo paso consiste en someter el valor del punto de cálculo, al umbral calculado.

Obtenemos de esta manera una imagen binarizada, utilizando un umbral adaptativo dependiente del valor de los puntos del entorno de cada punto de imagen.

Estudio de viabilidad hardware :

Podemos ver un esquema de bloques para una realización hardware que permite la computación en tiempo real, en la figura 3. Consta básicamente de memorias de almacenamiento, un multiplicador, un comparador y dos sumadores de alta velocidad. El almacenamiento de los valores de pixel se resuelve utilizando memorias estáticas de alta velocidad del tipo IDT71B024, y memorias FIFO, del tipo IDT72215A, para almacenar los datos de las columnas internas y externas al entorno del punto de calculo. El bloque de lógica de control, resuelve el flujo de datos entre operadores y memorias, incluyendo el control de número de puntos de entornos no completos por localizarse en los

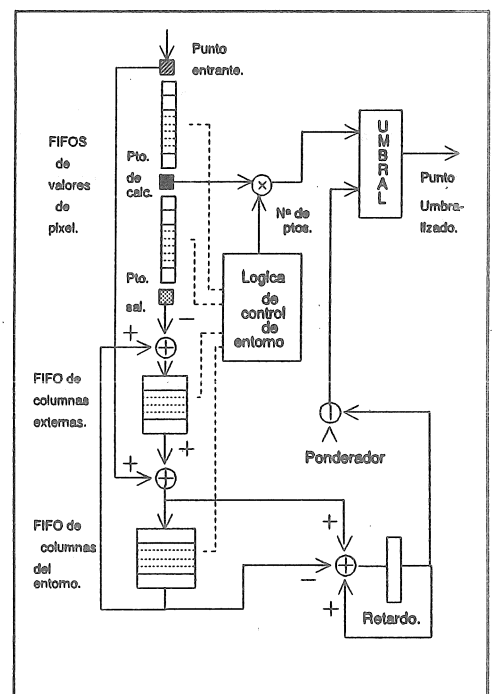


Fig. 3. Esquema de bloques HW.

bordes de imagen. El resto de las operaciones de suma y multiplicación se realizan de manera segmentada utilizando EPLD de ALTERA del tipo EP600 y EP900, al igual que el bloque de lógica de control.

Simulación :

Para la simulación del sistema se ha trabajado en un entorno compuesto por una cámara de televisión CCD, blanco y negro de norma CCIR, una placa comercial de digitalización de imagen de la casa Matrox, y un grupo de ordenadores tipo PC. El sistema se ha simulado mediante rutinas en C, y se ha valorado digitalizando varias imágenes significativas y sometiéndolas a proceso según el método y algoritmos propuestos. Los resultados obtenidos se comentan a continuación.

Resultados y Conclusiones :

Como muestra de los resultados obtenidos, exponemos aquí algunos ejemplos significativos.

Se ha sometido la imagen de la figura 4, sobre la que se imprime el histograma de la misma, a varios tipos de filtrado basados en el nivel de gris, obteniendo los resultados que comentamos a continuación.

- La fig. 5, muestra el resultado de someter la imagen a un umbral fijo en todos sus puntos. La elección de este umbral se ha realizado analizando el histograma. El umbral se fijo en el nivel de gris 135 (punto 1 de la fig. 4). Puede verse que conseguimos segmentar solo una zona de la imagen, siendo defectuosa la umbralización en zonas de distinto nivel de gris.

- La fig. 6, podemos ver el resultado de someter la imagen a un umbral del mismo tipo, pero con valor 176 (punto 2 de la fig. 4). Conseguimos segmentar otra zona de la imagen, pero perdemos gran cantidad de información en las zonas anteriormente bien segmentadas.

Un umbral de valor igual a la media de todos los valores de los puntos de imagen, da resultados mas uniformes pero de peor calidad, como puede verse en la figura 7.

Utilizando el algoritmo que proponemos, puede someterse la imagen a umbrales adaptativos, consiguiendo resultados como los que muestran, para distintos tamaños de entorno y tipo de umbral, las figuras 8, 9, 10.

- Fig. 8 . Umbral adaptativo:

Umbral = media de los valores del entorno de 20 x 20.

Se detecta la información en todas las zonas de imagen. En zonas de

niveles de gris muy similares, se detectan los puntos con pequeñas diferencias y el ruido.

- Fig. 9 . Umbral adaptativo:

Umbral = media de los valores del entorno de 60 x 60.

Los resultados son similares al caso anterior, pero puede apreciarse la diferencia de comportamiento en las fronteras.

- Fig. 10 . Umbral adaptativo:

Umbral = media de los valores del entorno de 20 x 20 + el mismo valor multiplicado por una cte.=0.05 en cada punto.

Con un umbral como este, adaptativo, pero mas exigente, eliminamos el ruido en las zonas donde no existe mucha diferencia de niveles de gris.

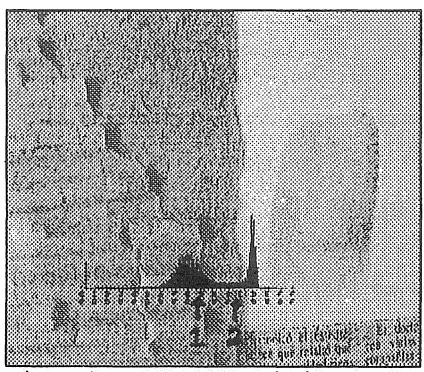


Fig. 4. Imagen original con histograma de la misma.

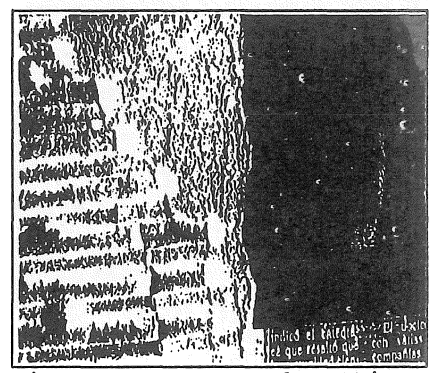


Fig. 5. No adaptativo. Umbral punto 1 fig. 4.

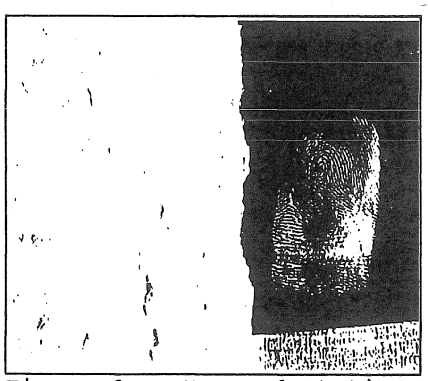


Fig. 6. No adaptativo. Umbral punto 2 fig. 4.

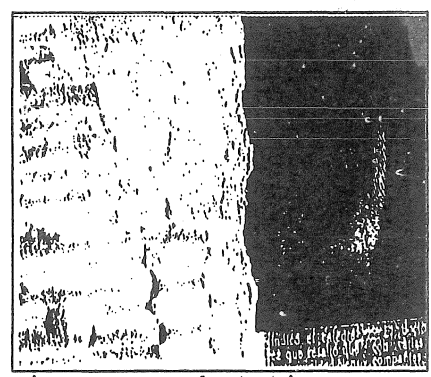


Fig. 7. No adaptativo. Basado en media de pantalla

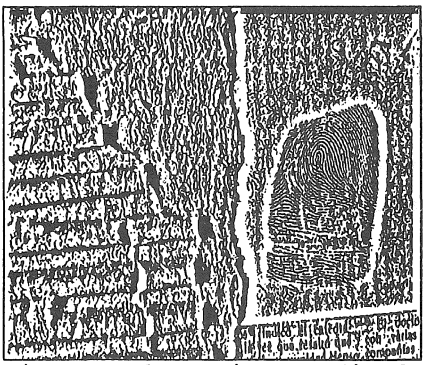


Fig. 8. Adaptativo. Media de entorno 20 x 20.

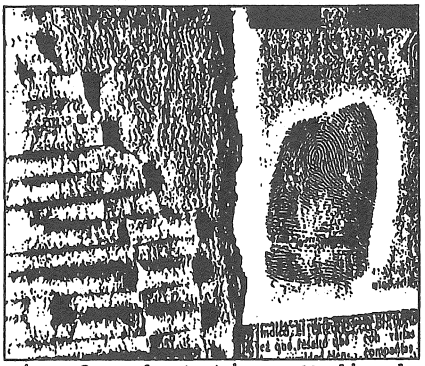


Fig. 9. Adaptativo. Media de entorno 60 x 60.

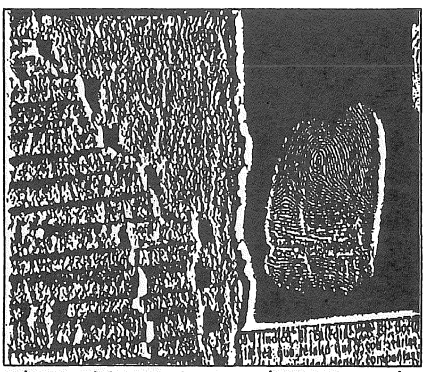


Fig. 10. Adaptativo. Media modificada entorno 20 x 20.

Podemos afirmar que el sistema de umbral adaptativo, basado en algoritmos de cómputo acumulado, es realizable mediante hardware para conseguir, en tiempo real, un valor de umbral en cada punto de imagen dependiente del valor de los puntos de su entorno.

Atendiendo a los resultados obtenidos en la fase de valoración, este tipo de umbralización consigue clasificar gran cantidad de información que no detectan los sistemas basados en umbrales fijos para toda la imagen. Puede destacarse que, el método que pondera la media para fijar el umbral, las dimensiones del entorno elegidas, y la situación del punto de cálculo dentro del entorno, juegan un papel importante en los resultados obtenidos, y pueden optimizarse para problemas concretos.

Este tipo de algoritmos de cómputo acumulable, realizado mediante un hardware similar al expuesto, puede ser utilizado para otras tareas referentes al proceso de imagen, como puedan ser el suavizado, o a otros campos de tratamiento de la señal que requieren el cálculo de la media de un entorno del punto en tiempo real.

- Agradecemos el apoyo económico prestado por los proyectos CICYT TIC-880315 y TIC-92-0136.

- Agradecemos a Luis Alonso Almansa Daimiel, la colaboración prestada en la fase de simulación.

Referencias :

[1] Ikram E. Abdou & William K. Pratt. "Quantitative Desing and Evaluation of Enhacement/Thersholding Edge Getectors".

[2] Ballard, Dana H. & Brown, Christopher M. . "Computer Visión". Prentice Hall (1982). Apartados 3.3.2 y 5.3.

[3] Kohler, Ralf . "A Segmentation System Based on Thresholding". Computer Graphics and Image Processing (1981).

[4] Robert J. Schalkoff. " Digital Image Processing and Computer Visión ". Wiley. Página 178 y siguientes.

[5] fu K. S., Gonzalez R. C. & Lee C. G. S.. "Robótica: Control, Detección, Visión e Inteligencia". Apartados 7.6.5. y 8.2.2.

[6] E. R. Davis. "Machine Vision: Theory, Algorithms, Practicalities". Academic Press. New York. Apartados 4.3 y 4.4.

The Binary-Radix Bucket-Region-Directory: A Simple New Directory for the Grid File

Ulrich Finke, Klaus Hinrichs, Ludger Becker
FB 15 - Informatik, Westfälische Wilhelms - Universität,
D - 4400 Münster, Germany

Abstract

The grid file introduced by Nievergelt et al. in [1] is a dynamic, symmetric multikey file structure. It organizes highly dynamic sets of multidimensional data on secondary storage in such a way that different types of queries can be performed using few disk accesses. We present the new BR^2-directory structure for the grid file. The BR^2-directory is implemented by a tree. It grows linearly with the number of data buckets and reduces both the CPU-costs for directory operations and the I/O costs for directory pages.

Keywords: File structures, GridFile, Directory structures, Buddy system, Multikey searching, Splitting Buckets, Merging Buckets, Dynamic storage allocation

1 Introduction

The grid file partitions the k-dimensional data space $D=d_1 \times d_2 \times \ldots \times d_k$ by an orthogonal grid. d_i $(1 \le i \le k)$ denotes the domain underlying dimension i. The grid is defined by (k-1)-dimensional hyperplanes. Each hyperplane is determined by a data value in one dimension, called a boundary. The boundaries of each dimension are represented by a scale. Two successive boundaries in a scale determine a binary radix interval. A new boundary is generated by halving such a binary radix interval (Fig. 1).

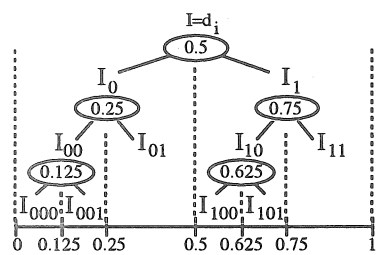


Fig. 1

Each cell of the grid is assigned to a physical disk block on secondary storage, called a bucket. All data points which lie in a grid cell can be found in the disk block assigned to this grid cell. A directory structure represents and manages the assignments of grid cells to buckets. The mapping is surjective: several grid cells may share a bucket. The grid cells covered by a bucket form the bucket's region (Fig. 2). The mapping has to adhere to the following rules which determine the buddy system:

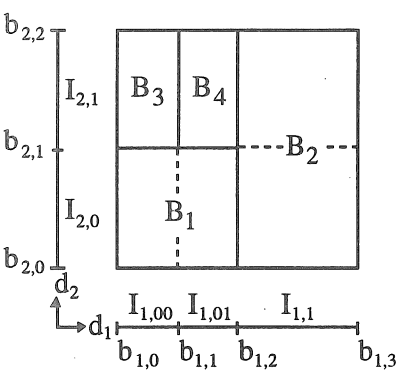


Fig. 2

- The bucket region must be a k-dimensional rectangular box:
  $B_n = I_1 \times I_2 \times \ldots \times I_k$, with $\forall_{j \in \{1 \ldots k\}}\ I_j \subseteq d_j$.
- Each interval I_j is a binary-radix interval.

Insertions and deletions of data points modify the grid dynamically. Insertions may lead to splitting of

overflowing buckets. If a bucket region consists of two or more grid cells it can be split along a dividing hyperplane. Otherwise a new boundary must be created by halving a bucket region interval I_j. The bucket-region-condition

$$\max\left(\frac{|d_1|}{|I_1|},\frac{|d_2|}{|I_2|},\ldots,\frac{|d_k|}{|I_k|}\right) \leq 2 \min\left(\frac{|d_1|}{|I_1|},\frac{|d_2|}{|I_2|},\ldots,\frac{|d_k|}{|I_k|}\right)$$

ensures that almost "quadratic" bucket regions are generated (heuristic). $|d_j|$ denotes the "length" of domain d_j, $|I_j|$ denotes the length of interval I_j.

Deletions of data points may lead to merging of underflowing buckets. After merging two buckets it may become necessary to remove a boundary, i.e. a row of the grid, which is no longer used. Merging is not simply the reversal of the split process. The buddy system may allow an underflowing bucket to merge with one of k potential candidates.

[1] left open some important questions such as the implementation of the scales and the grid directory. An obvious approach is the *grid array* [2], which represents the grid by a unique multidimensional dynamic array. Each grid cell corresponds to an array element. However, inserting and removing boundaries, i.e. inserting or removing a row in any dimension, are time consuming operations since virtually all array elements have to be moved. Furthermore the grid array may grow superlinearly with the number of buckets for non uniform data distributions [3].

The *region representation* [4] stores all bucket regions together with the bucket addresses in a linear list. This approach guarantees that the directory grows linearly with the number of buckets. Both the explicit representation of bucket regions and the list implementation lead to time consuming search- and management-operations.

The objective of our work was to develop a new kind of directory which guarantees a linear growth with the number of buckets, ought to be easy to implement and supports efficient directory management operations.

2 The BR^2-directory

Each boundary of a scale divides the data space into two parts. The BR^2-directory represents the subdivision of the data space by a tree-like structure.

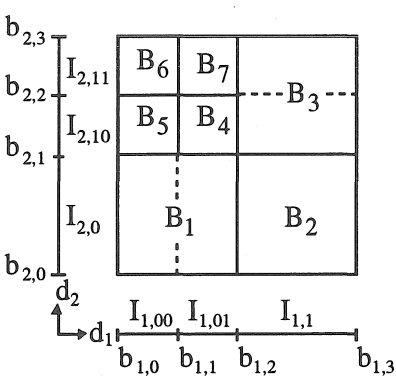


Fig. 3

2.1 Construction of the BR^2-tree

We explain the construction of a BR^2-directory for the two-dimensional partitioning of the data space given in Fig. 3. Two scales each containing four boundaries $b_{j,0} \ldots b_{j,3}$ determine the grid consisting of nine cells. The binary sequence S determines how an interval $I_{j,S}$ is obtained by subsequent bisection from d_j: A zero (0) means selection of the lower interval, a one (1) selection of the upper interval. $B_1 \ldots B_7$ represent the buckets. Except for B_1 and B_3, they only cover one grid cell.

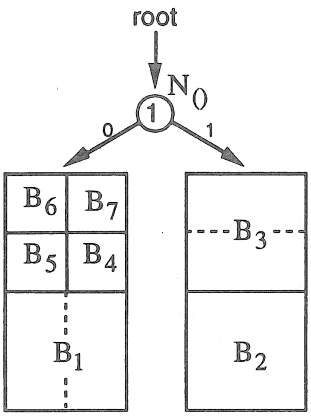


Fig. 4

Each boundary partitions the data space into two. For a grid file which has been generated by insertions only the buddy system guarantees the existence of at least one boundary which does not intersect any bucket region, e.g. $b_{1,2}$ and $b_{2,1}$ in Fig. 3. We select boundary $b_{1,2}$ which separates the buckets B_1, B_4...B_7 from the buckets B_2 and B_3 (Fig. 4). Therefore the root node N_0 of the BR^2-tree obtains an entry for the dimension of $b_{1,2}$, i.e. node dimension $d_{N_0} = 1$.

We continue by selecting boundaries and represent them by new nodes, which contain their dimension as an entry. Fig. 5 shows one of the possible BR^2-trees for our example. The pointers to the leaves are the identifiers of the buckets on disk.

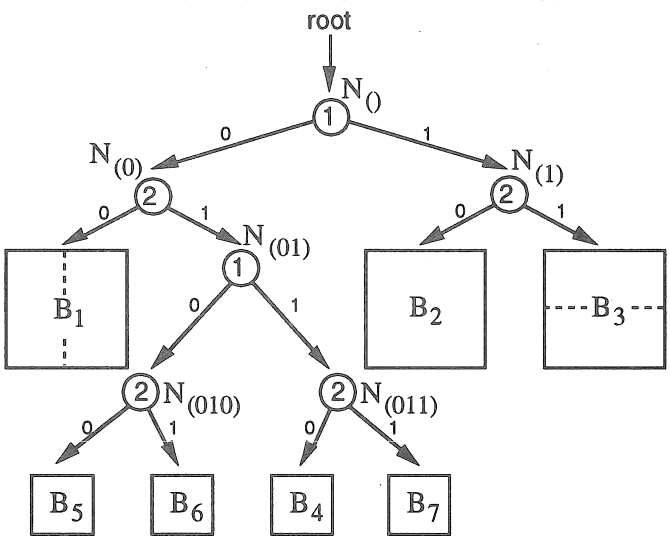


Fig. 5

2.2 Properties of BR^2-trees

Let n be the number of buckets. We summarize some properties of BR^2-trees:

- In binary trees the number of internal nodes is equal to the number of leaves minus one. Therefore the size of the BR^2-tree grows linearly in n: $\Theta_{space}(n)$.
- Each internal node corresponds to an unique boundary in one of the scales. A boundary may be assigned to more than one internal node.
- Each node N (internal node or leaf) represents a rectangular region D_N of the data space; this region is the Cartesian product of binary-radix intervals represented by the scales. The depth of a node or leaf determines the size of this region.

- Each internal node or leaf owns a unique path, which can be described by a bit-sequence S. In Fig. 5 these bit sequences S form the node identifiers $N_{(S)}$. If S_1 is the bit-sequence of a node N_1 and S_2 the bit-sequence of a node or leaf N_2, then:
  S_1 is prefix of $S_2 \Leftrightarrow D_{N_1} \supseteq D_{N_2}$.
- The BR^2-representation of the partitioning is ambiguous. In most cases there exist several different equivalent BR^2-trees, e.g. four trees for the example of Fig. 3.

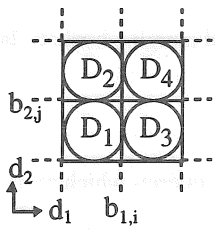


Fig. 6

Each tree can be obtained from any equivalent tree by performing a number of transformations. Fig. 6 shows a subdivision of one part of the data space. $D_1 \ldots D_4$ could be bucket regions or further be subdivided. The two equivalent BR^2-subtrees representing this subdivision are shown in Fig. 7.

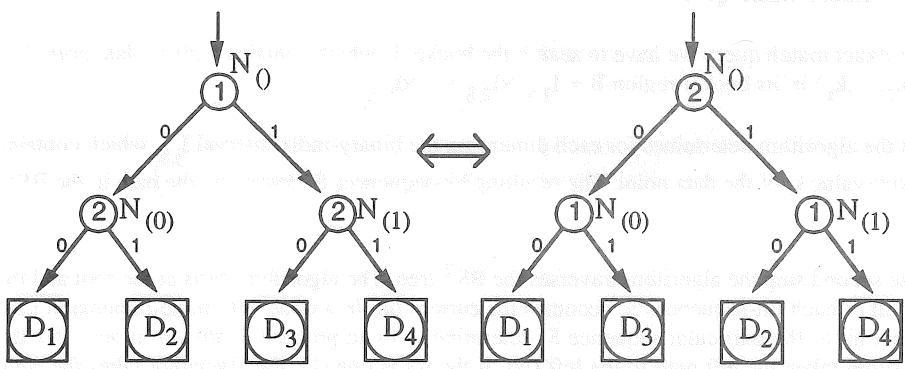


Fig. 7

Each subtree can be obtained from the other subtree by exchanging the node-dimension of the node N_0 with the node-dimension of sons $N_{(0)}$ and $N_{(1)}$ and exchanging the references of D_2 and D_3. Such a d-transformation can always be carried out if the two direct sons of a node have the same node-dimension which must be different from the node dimension of the father.

The dimensions which can appear in a node N after any d-transformations have been performed in the subtree rooted at N are called $PND_N \subseteq \{1,\ldots,k\}$, the set of possible node dimensions of N. PND_N can be determined recursively:

- $PND_{Leaf} = \varnothing$.
- If $N_{(S)}$ is an internal node with node dimension $d_{N_{(S)}}$ and direct sons $N_{(S0)}$ and $N_{(S1)}$, then
  $PND_{N_{(S)}} = (PND_{N_{(S0)}} \cap PND_{N_{(S1)}}) \cup \{ d_{N_{(S)}} \}$.

For a node N the set PND_N does not change when any d-transformations are performed in the subtree rooted at N. The complexity to determine PND_N for all nodes N of a given BR^2-tree is: $\Theta_{Time}(n)$.

In an implementation of the BR^2-tree each node can be represented by 8 bytes; three bytes for each pointer to the sons and two bytes for additional information, e.g. the node dimension. Since the identifiers for buckets are stored in their father nodes we need $8(n-1)$ bytes for n buckets. This is less than is needed in the grid array representation (even under uniform data distribution) and the region representation.

2.3 Operations on BR^2-trees

Fundamental operations on directory structures are searching, splitting and merging buckets. In the following we discuss these operations.

2.3.1 Searching buckets

Directory representations have to support efficient exact match and range queries which return references to buckets containing the records to be searched for.

a) Exact match query

In an exact match query we have to search the bucket B which contains a given data point $P = (k_1,k_2,\ldots,k_k)$ in its bucket region $B = I_{1,S_1} \times I_{2,S_2} \times \ldots \times I_{k,S_k}$.

First the algorithm determines for each dimension the binary-radix interval I_{j,S_j} which contains the key-value k_j of the data point. The resulting bit-sequences S_j determine the path in the BR^2-tree.

In the second step the algorithm traverses the BR^2-tree: The algorithm starts at the root and the first bit of each bit-sequences S_j becomes the current bit. In a node with node-dimension j the current bit of the particular sequence S_j determines how to proceed. If this bit is zero (0), the algorithm takes the left path to the left son, if the bit is one (1), the algorithm takes the right path. Then the next bit from sequence S_j becomes the current bit. The algorithm terminates as soon as a leaf is reached.

The costs for this search in the BR^2-directory and the scales are for uniform data distribution $\Theta_{Time}(\log n)$.

b) Range query

The range query is formally defined as follows: Search all buckets B whose bucket regions B intersect a given query range $R = (I_1, \ldots, I_k)$ where I_j is an interval in the range of dimension d_j. The query is again performed in two steps:

- Determine for each interval $I_j = [l_j, u_j]$ two bit sequences L_j and U_j for the lower and upper bound of I_j in the binary-radix tree for dimension d_j.
- Traverse the BR^2-directory.

Remember each node N of the BR^2-directory is represented by an unique bit sequence S_N. For example the representation (or path) to bucket B_4 in Fig. 5 is $S_{B_4} = 1_0 \cdot 2_1 \cdot 1_1 \cdot 2_0$. The corresponding part of the data space is the cartesian product of k binary radix intervals which can again be described by k unique bit sequences. Let $S_{N,j}$ be the bit sequence for dimension j (e.g. $S_{B_4,2}=2_1 \cdot 2_0$).

A node N is relevant for the range query if all sequences $S_{N,j}$ $(1 \le j \le k)$ satisfy $L_j \le_s S_{N,j} \le_s U_j$ where $\le_s$ is the lexicographic order on bit strings. Starting at the root of the tree, we distinguish the following cases at a node N which has node dimension j (Fig. 8):

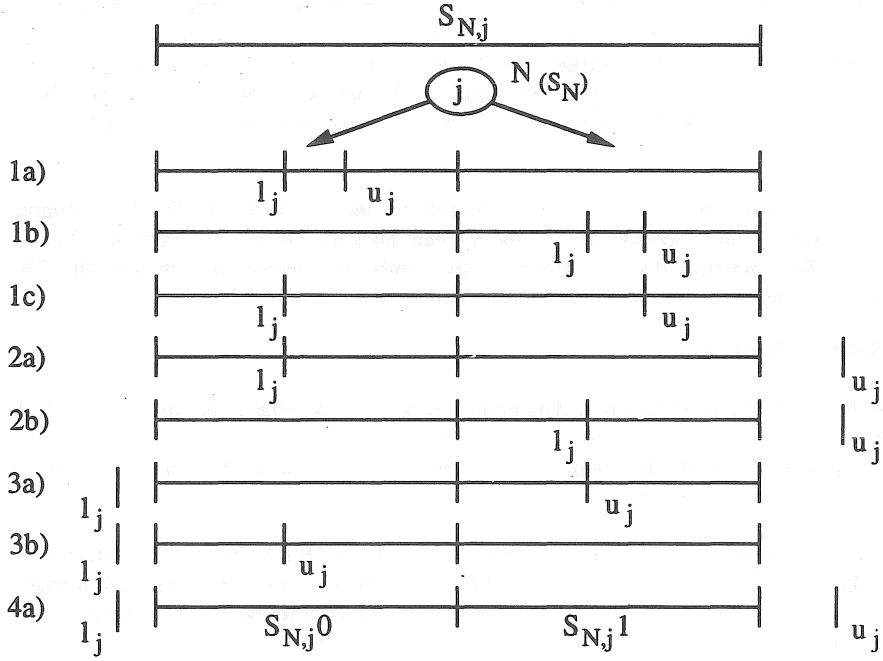


Fig. 8

1) $S_{N,j}$ is prefix of L_j and U_j, i.e. the binary radix interval corresponding to the region of node N in dimension j contains both the lower bound l_j and the upper bound u_j. If the current bits of the sequences L_j and U_j are both 0, we have to continue with the left son (1a), if the current bits are both 1, we have to continue with the right son (1b). If the current bits are different we have to visit both sons (1c).

2) $S_{N,j}$ is prefix of L_j, but not of U_j, i.e. the binary radix interval corresponding to the region of node N in dimension j contains the lower bound l_j. If the current bit of sequence L_j is 0 we have to visit both sons of N (2a). Otherwise only the right son is considered (2b).

3) $S_{N,j}$ is prefix of U_j, but not of L_j, i.e. the binary radix interval corresponding to the region of node N in dimension j contains the upper bound u_j. If the current bit of sequence U_j is 1 we have to visit both sons of N (3a). Otherwise only the left son is considered (3b).

4) $S_{N,j}$ is neither prefix of L_j nor of U_j, i.e. the binary radix interval corresponding to the region of node N in dimension j contains neither the lower bound l_j nor the upper bound u_j. Therefore we have to visit both sons (4a).

It is not necessary to construct the bit sequences of a visited node because the prefix properties can be inherited from the father. It is sufficient to introduce 2k status variables to describe the prefix properties of each bit sequence. The other tests in the algorithm are simple bit comparisons.

The BR^2-directory has the advantage that every bucket in the query range is returned exactly once. In the grid array representation a bucket may cover more than one grid cell and hence it may be returned more than once during range queries. Therefore we have to decide for each bucket whether it has already been visited. For this test we must construct its bucket region, and determine whether this region extends into a part of the data space which has already been processed. Constructing the bucket region from the grid array representation is a CPU-time consuming task.

The BR^2-directory is superior to the region directory for the following reasons. More simple and less comparisons are necessary to find the relevant buckets. The space to represent bucket regions in the BR^2-directory does not depend on the number of dimensions of the grid file. The overall space requirements are higher for the region directory.

2.3.2 Splitting buckets

A bucket B can overflow when a new data point $P = (k_1,k_2,\ldots,k_k)$ has to be inserted into B. If B overflows it has to be split and new disk blocks have to be allocated. We assume that the bit sequences S_j determining the BR^2-path to B have been computed as in the exact match query. Then the split can be performed as follows:

First the split dimension, i.e. the dimension according to which B is split, must be determined. If the bucket region covers more than one grid cell B can be split along an existing boundary. The existence of such a boundary can be determined by examining the bit sequences S'_j remaining after the traversal of the BR^2-tree. B can be split according to dimension j if S'_j is not the empty.

If no such dimension exists no existing boundary divides B's region. In this case we select a split dimension j and a boundary b_j as has been explained in section 1 (bucket-region-condition). We insert the boundary into the particular radix tree of scale j. In any case we can now split B by allocating a new internal node N and a new bucket B'. The old father of B now points to N. The node dimension of N is j, the left son is B, the right son is B'. All data points stored in B are distributed among B and B' depending on their key value in dimension j.

The insertion of a new boundary in a scale costs $O_{Time}(\log n)$ for uniform data distribution. The remaining costs of a split are $\Theta_{Time}(1)$.

2.3.3 Merging buckets

The grid file has to adapt gracefully to a varying distribution of records. Deletions of data points may lead to almost empty data buckets. In order to achieve a high storage utilization buckets may be merged if possible. The buddy system may allow an underflowing bucket to merge with one of k potential candidates. In a grid file of three or more dimensions merging of buckets according to the buddy system may lead to deadlocks (Fig. 9), i.e. buckets are generated which prevent each other from further merging since their resulting bucket region would not be rectangular.

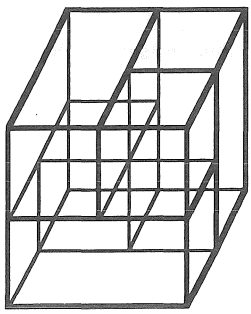


Fig. 9

Both the grid array and the region representation can represent such a deadlock configuration, but there exists no BR^2-tree for such a space partitioning. In general each space partitioning represented by a BR^2-tree is deadlock-free: In a BR^2-tree there exists at least one internal node which has two leaves as direct sons. Thus in each BR^2-directory we can find at least two buckets which can be merged if they become almost empty.

Some necessary criteria for finding a merge candidate B_2 for an underflowing bucket B_1 are:

(1) B_2 must have the same depth in the BR^2-tree.

(2) B_1 and B_2 can be merged only if their joint occupancy is well below the bucket capacity.

(3) If we have found a candidate B_2 according to (1) and (2), then B_1 and B_2 must have a common father node FATHER. We describe the path to a node N (internal node or leaf) by a sequence S_N which contains for each path node its dimension and the direction of traversal (see 2.3.1.b for an example).

Furthermore we denote the path from node N_1 to N_2 with $S_{N_1 \to N_2}$. Then the following holds for S_{B_1} and S_{B_2} (Fig. 10):

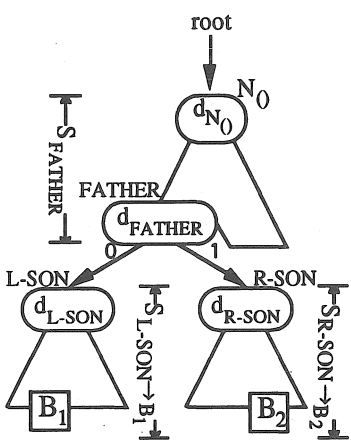


Fig. 10

$$S_{B_1} = S_{FATHER} \cdot (d_{FATHER})_0 \cdot S_{L\text{-}SON \to B_1}$$

$$S_{B_2} = S_{FATHER} \cdot (d_{FATHER})_1 \cdot S_{R\text{-}SON \to B_2}$$

The path branches out at the common father node. Since for a merge the buckets must be neighbors in data space, the paths $S_{L\text{-}SON \to B_1}$ and $S_{R\text{-}SON \to B_2}$ must satisfy the following conditions:

- The paths are not allowed to contain the father's node dimension d_{FATHER}.
- Let $S_{N,j}$ denote the path in dimension j; then the following condition must be true:

$$\forall_{j \in \{1 \ldots k\}} S_{L\text{-}SON \to B_1, j} = S_{R\text{-}SON \to B_2, j}$$

(4) $|PND_{FATHER}| \geq 2.$

If a bucket meets these conditions a sufficient check can be made:

(M1) [Select dimension]
Select a dimension j from PND_{FATHER}.

(M2) [Transform subtree]
Transform the subtree rooted at FATHER with d-transformations, such that j becomes the new node dimension of FATHER. The buckets B_1 and B_2 are descendants of their new father node L-SON or R-SON.

(M3) [Look for new father]
Make the particular son to the new father-node:
Either L-SON → FATHER or R-SON → FATHER.

(M4) [Merge buckets]
If B_1 and B_2 are direct sons of the new father, then merge the buckets and terminate.

(M5) [Loop or termination]
Determine PND_{FATHER}. If PND_{FATHER} satisfies condition (4) then continue with (M1), else terminate.

If the algorithm terminates in (M4), the buckets can be merged by copying all data points from B_2 to B_1 and replacing the father-node by B_1. If possible a search for other canditates follows a termination in (M5).

The total costs for finding all merge candidates of a given bucket are $O_{Time}(2^k)$ for uniform data distribution.

3 Implementation issues

We have presented the basic idea of the BR^2-directory representation. To improve performance we introduce some additional concepts:

2-level directory

Since in general a directory representation does not fit in main memory Hinrichs [2] proposed the 2-level directory (Fig. 11). The first level is the root directory, a scaled down version of the grid directory. The boundaries which determine the grid corresponding to the root (and page) directory(ies) are contained in the scales. An element of the root directory is a pointer to a directory page on secondary storage which contains the corresponding part of the grid directory. The elements of the subdirectories are references to data buckets. The constraints defined for the grid directory apply to the root directory as well as the subdirectories. In [2] the root directory and the subdirectories are represented by grid arrays. It is obvious that the 2-level technique can also be applied to the BR^2-directory representation.

Empty buckets

Since for non uniform data distributions the grid file can contain many empty buckets, the storage utilization may degenerate if all these empty buckets are allocated on disk. This degeneration is avoided by replacing each reference to a bucket B which becomes empty by a

special 'empty bucket'-reference - similar to a NIL-pointer in dynamic data structures. The bucket B can now be deallocated.

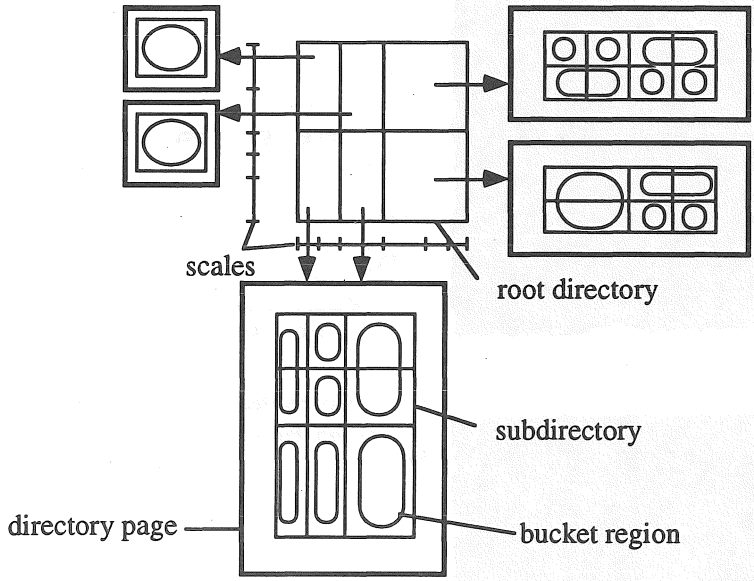


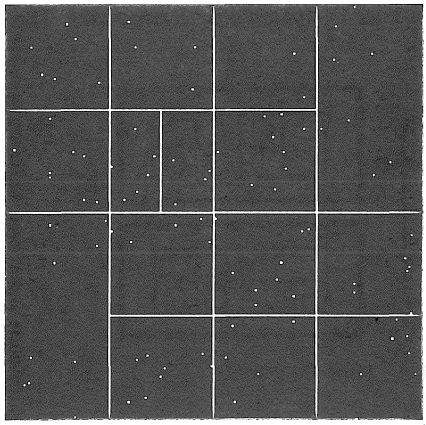
Fig. 11

Buffer management

The performance of operations on access methods is increased by using a part of the available main memory as a buffer. For a 2-level directory this buffer should contain the root directory and the scales, because every directory operation must access these parts. The remaining memory can be used for storing pages and buckets; the distribution of this part of the system buffer should be managed by a heuristic strategy like LRU. However, efficient operations on access methods require that a set of pages and buckets can be locked by operations, i.e. they cannot be removed from the system buffer.

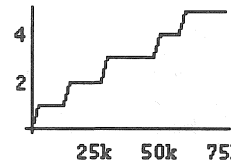
4 Tests

For tests we have implemented a BR^2-directory based grid file together with a test system called 'InsideGrid' which can be used to generate test data and to study the behavior of the directory. In the following we show the growth of the directory in the data pages (512 byte) for different data distributions. To get interesting results the page and bucket size is one data page and the length of a record is 50 bytes, so every bucket can store up to 9 records. Better results can be expected for larger pages and buckets. During every test 75000 records are inserted in a GridFile. The first, second and third diagram show the number of data pages used for the root directory, the directory pages and buckets. The number of empty bucket references (not allocated buckets!) is shown in the fourth diagram. For two dimensional data distributions an example of the corresponding partitioning of the data space is shown.

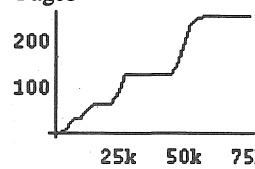


I. **Uniform data distribution:**

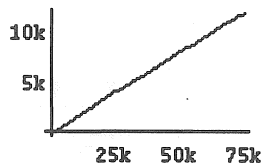
Root



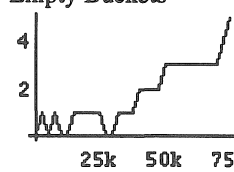
Pages

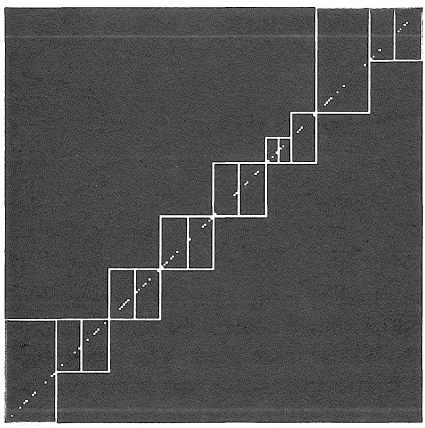


Buckets



Empty Buckets

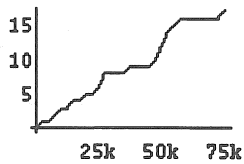




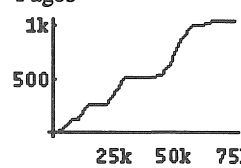
II. **Correlated data distribution:**

a. Diagonal through data space

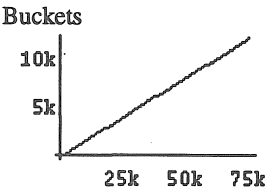
Root



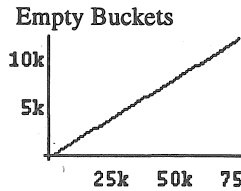
Pages

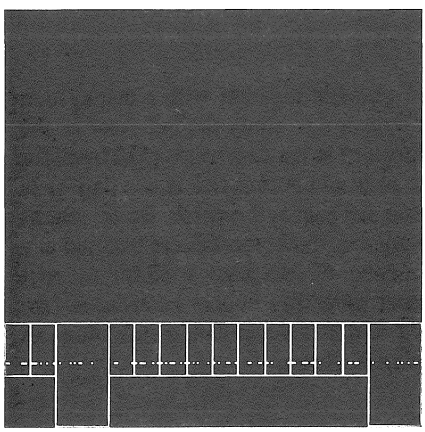


Buckets



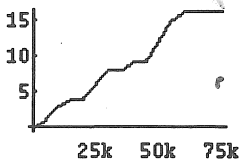
Empty Buckets



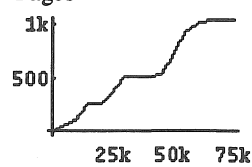


b. Second key is constant

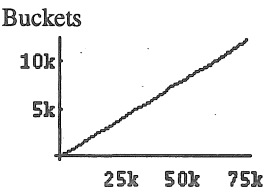
Root



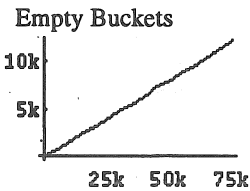
Pages

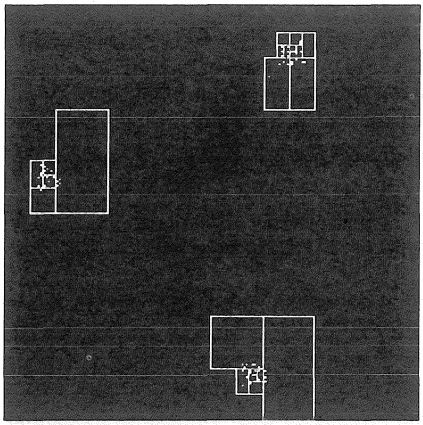


Buckets

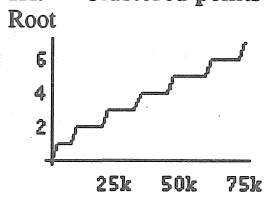


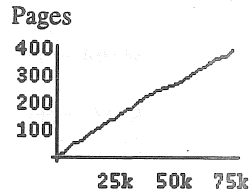
Empty Buckets

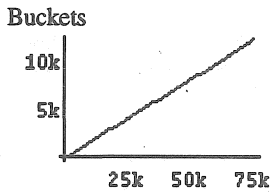


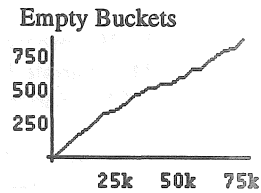


III. Clustered points:



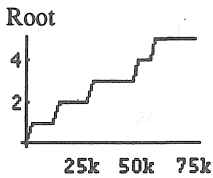


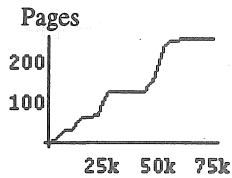


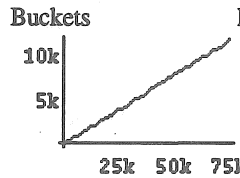


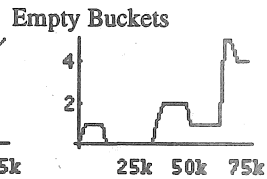
IV. **Data distribution by storing extended objects: (4-dimensional)**

a. Center transformation:

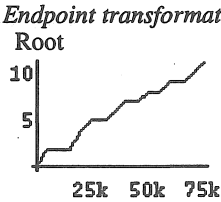


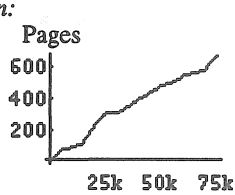


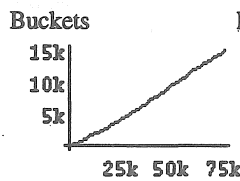


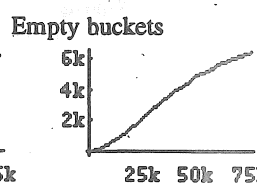


b. Endpoint transformation:









5 Conclusions

In this paper we have presented the BR^2-representation for the directory of a grid file. This representation guarantees that the directory grows linearly with the number of data buckets. Besides the algorithms for exact match and range queries we have developed evaluation strategies for other queries using grid files with a BR^2-representation of the directory, e.g. region and join queries [5], [6].

Since all elements of a BR^2-directory are simple binary trees, it is straightforward to apply techniques developed for concurrent operations on B-trees and other dynamic data structures [7], [8], [9] to a grid file with BR^2-directory.

References

[1] J.Nievergelt, H. Hinterberger, K.C. Sevcik: The Grid File: An Adaptable, Symmetric Multikey File Structure, ACM TODS 9, 1 (1984), 38 - 71.

[2] K. Hinrichs: Implementation of the Grid File: Design, Concepts and Experience, BIT 25 (1985), 569 - 592.

[3] M. Regnier: Analysis of Grid File Algorithms, BIT 25 (1985), 335 - 357.

[4] H. Hinterberger: Data Density: A Powerful Abstraction to Manage and Analyze Multivariate Data, Dissertation No. 8330, Eidgenössische Technische Hochschule (ETH), Zürich, 1987.

[5] L. Becker, U. Finke, K. Hinrichs: Efficient Join Processing with Multidimensional File Structures, Informatik Bericht Nr. 90 - 03, Universität-GH-Siegen.

[6] L. Becker, K. Hinrichs, U. Finke: A Cost Model for Query Processing with Grid Files, Informatik Bericht Nr. 91 - 01, Universität-GH-Siegen.

[7] C.S. Ellis: Concurrency in extendible hashing, Information Systems (1988), Vol. 13, No. 1, 97-109.

[8] Y. Kwong, D. Wood: Method for concurrency in B-trees, IEEE-TOSE (1982), Vol. 8, No. 3, 211-223.

[9] Shasha D., Goodman N.: Concurrent search structure algorithms, ACM-TODS (1988), Vol. 13, No. 1, 53-90

PROBLEMATICA DE LA ADQUISICION INDUCTIVA DE CONOCIMIENTO A PARTIR DE EJEMPLOS

Gabriel Fiol
Dpt. de Ciències Matemàtiques i Informàtica
Universidad de las Islas Baleares
E-mail: dmigfr0@ps.uib.es

José Miró-Nicolau
Dpt. de Ciències Matemàtiques i Informàtica
Universidad de las Islas Baleares
E-mail: dmijmn0@ps.uib.es

1. INTRODUCCION

La *adquisición de conocimiento* es uno de los campos de la Inteligencia Artificial que más atención está recibiendo en la actualidad. Un sistema de adquisición de conocimiento es un subsistema de un sistema de aprendizaje encargado de la transferencia de conocimiento desde el *entorno* a la *base de conocimientos*. En la figura 1 se ilustra un modelo general de un sistema de adquisición de conocimiento.

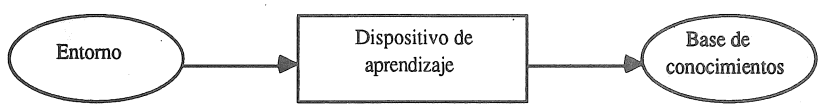


Figura 1. Modelo general de un sistema de adquisición de conocimiento

Las elipses de la figura 1 representan elementos de conocimiento mientras que los rectángulos representan procedimientos y las flechas el flujo de la información, de manera que el entorno suministra alguna información al *dispositivo de aprendizaje*, el cual hace uso de dicha información para perfeccionar los conocimientos almacenados en la base de conocimientos.

Uno de los tipos de adquisición de conocimiento más estudiados en la actualidad es la adquisición inductiva de conocimiento a partir de ejemplos o elementos específicos de conocimiento. En estos sistemas, el entorno está formado por una colección de ejemplos o elementos específicos de conocimiento referentes a algún concepto. La función del dispositivo de aprendizaje consiste en generalizar la información proporcionada por los ejemplos con objeto de encontrar alguna descripción a alto nivel del concepto.

La problemática de la adquisición inductiva de conocimiento se apoya entre otros, en dos aspectos fundamentales:

- En el establecimiento de métodos de generalización de la información proporcionada por los ejemplos.
- En la mecanización del proceso de diseño de sistemas de adquisición de conocimiento, de manera que se permita abordar la solución de cualquier problema relacionado con el tema.

El estudio sobre la adquisición inductiva de conocimiento se ha centrado mayoritariamente en la búsqueda de métodos de generalización, siendo numerosos los trabajos sobre el tema, entre los que destacan: *El método AQ11*, propuesto en [5], que genera una descripción general de un conjunto formado por ejemplos positivos, en contraste con otro conjunto de ejemplos negativos. Para ello se basa en las propiedades comunes de los elementos del primer conjunto que no posee ningún elemento del segundo conjunto. El resultado viene dado en forma de reglas de decisión. *El método ID3 (Iterative Dicotomizer 3)*, descrito en [10], genera un *árbol de decisión* a partir de un conjunto de ejemplos caracterizados por un conjunto de atributos, haciendo uso de la función entropía definida por Shanon [11] para seleccionar los atributos en cada nivel del árbol. Este método se ha caracterizado por la elegancia de la idea central y por su fácil implementación. El concepto de *Rough Set* [7, 8, 9] trata del estudio sobre descripciones aproximadas de subconjuntos y surgió como alternativa a otras investigaciones en torno al tema, tales como los *Fuzzy Sets*. Su influencia ha sido notable, dando lugar a toda un área de investigación en torno al tema. En [1], [2] y [3] se presenta un nuevo enfoque al tema, basado en la teoria discutida en [6] que sumarizamos en la sección 2.

No ha ocurrido lo mismo en cuanto a la investigación en torno a las metodologias de diseño de sistemas de adquisición inductiva de conocimiento. Ello ha sido debido en gran parte a la disparidad de los métodos establecidos y a la falta de profundización en determinados aspectos, especialmente los referentes al entorno del sistema. En la sección 3 del presente trabajo se exponen los aspectos generales concernientes al diseño de sistemas de adquisición inductiva de conocimiento, basados en la teoría presentada en [6].

2. PRELIMINARES

El problema estudiado puede describirse formalmente como sigue:

Sea $D = \{d_1, d_2, \ldots, d_m\}$ un conjunto de ejemplos o elementos concretos de conocimiento definido extensionalmente y $R = \{r_1, r_2, \ldots, r_n\}$ un conjunto de atributos (binarios o multivaluados), también definidos extensionalmente, tal que cada $d_i \in D$ tiene asociado un valor respecto a cada uno de los atributos. Dados los subconjuntos D_i, $D \supseteq D_i$, $i = 1, 2, \ldots, k$, también definidos por extensión, $D_i = \{d_a, d_b, \ldots, d_c\}$, se desea encontrar una descripción intensional P_i, $i = 1, 2, \ldots, k$, de cada uno de los subconjuntos, $D_i = \{d_j \in D / P_i\}$, $i = 1, 2, \ldots, k$, que sea óptima según algún criterio.

Las descripciones P_i deberán ser expresadas en términos de un subconjunto R_b, $R \supseteq R_b$, de atributos. Debe encontrarse, pues, un subconjunto de atributos óptimo en algún sentido tal que P_i pueda expresarse como función de ellos.

La información sobre los ejemplos será almacenada en la *Tabla Objeto Atributo (OAT)*. La OAT se define como un 4-tuple <D, R, V, F>, donde D es el conjunto de ejemplos o elementos concretos de conocimiento, R es el conjunto de atributos, $V = \{V_1, V_2, \ldots, V_n\}$ es un conjunto

de conjuntos: cada V_i es el dominio de valores que el atributo r_i puede adoptar, y $F = \{f_1, f_2, \ldots, f_n\}$ es un conjunto de funciones que definen extensionalmente los valores que los elementos de D adoptan respecto a los atributos de R, de forma que la función f_j define extensionalmente los valores que cada $d_k \in D$ adopta respecto al atributo $r_j \in R$, $f_j: Dx\{r_j\} \rightarrow V_j$. Se dice que cada $d_i \in D$ viene descrito por su correspondiente n-epla de valores de los atributos de la OAT.

Se introduce un nuevo objeto que llamamos *Tabla de Definición de Subconjuntos (SDT)*. La SDT describe, en un formato compatible con el de la OAT, los subconjuntos que desean describirse intensionalmente. Así, la SDT se define como un 4-tuple <D, R', V', F'>, donde D tiene el mismo significado que su homónimo de la OAT, $R'=\{r_{n+1}, r_{n+2}, \ldots, r_{n+w}\}$ es un conjunto de atributos y $V'=\{V'_1, V'_2, \ldots, V'_w\}$ es un conjunto de conjuntos, de forma que V'_i es el dominio de valores del atributo r_{n+i}. Cada valor $v_i^j \in V'_i$, $i = 1, \ldots w$, $j \leq m$, define una clase de equivalencia de D: la clase de los elementos de D que poseen el valor v_i^j respecto al atributo r_{n+i}, definiendo V'_i un conjunto de clases de equivalencia de D. $F'=\{f_{n+1}, f_{n+2}, \ldots, f_{n+w}\}$ es un conjunto de funciones, cada una de las cuales indica la clase de equivalencia a la que pertenecen los elementos de D respecto a los valores de un determinado atributo, es decir, f_{n+j}: $Dx\{r_{n+j}\} \rightarrow V'_j$. Obsérvese como los r_{n+j}, $i = 1, 2, \ldots, w$, tienen la misma naturaleza conceptual que los r_i de la OAT, pues f_{n+j} asigna a cada elemento de D el valor de una determinada cualidad. Cada w-epla de valores de los atributos de la SDT define las clases de equivalencia (subconjuntos) a los que pertenece su correspondiente ejemplo en D.

Por cuestiones de claridad llamamos *atributos condición* a los atributos de R y *atributos consecuencia* a los atributos de R'. Además, en la figura 2 se realiza una ilustración gráfica conjunta de la OAT y la SDT; este nuevo objeto será llamado OAT ampliada. Donde $t_i^j \in V_j$, $1 \leq j \leq n$, $1 \leq i \leq m$ y $t_i^{n+k} \in V'_k$, $1 \leq k \leq w$.

El problema mencionado al principio de esta sección puede ser redefinido como sigue: "Encontrar una descripción óptima de cada uno de los atributos consecuencia r_{n+i}, $i = 1, 2, \ldots, w$, expresada en términos de un subconjunto de atributos condición R_b, $R \supseteq R_b$".

Un concepto esencial en torno al cual gira el enfoque del tema es el de *base de atributos* (o simplemente *base*) [1], [6]. Una base de un conjunto de atributos R respecto a otro conjunto de atributos R', es un subconjunto de R que proporciona información lo suficientemente adecuada para describir los atributos de R'. Formalmente, el término 'información adecuada' se aplica a aquella información que permite describir los atributos de R' sin ningún tipo de confusión (contradicción). Por tanto, mediante una base de atributos de R respecto a R', es posible describir los atributos de R' sin confusiones.

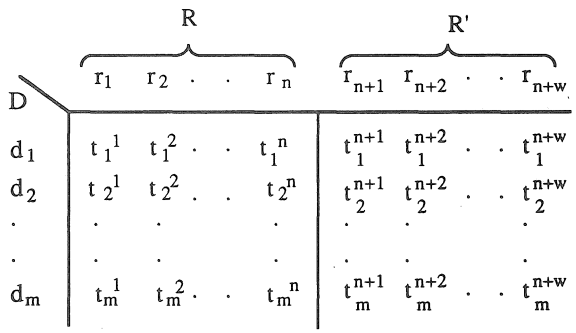


Figura 2. Ilustración gráfica de la OAT ampliada

3. ASPECTOS GENERALES DE LOS SISTEMAS DE ADQUISICION INDUCTIVA DE CONOCIMIENTO A PARTIR DE EJEMPLOS

El diseño de un sistema de adquisición inductiva de concimiento a partir de ejemplos se basa en el desarrollo de cada una de las etapas de que consta el sistema, tal como se ilustra en la figura 1. Por tanto, resulta fundamental para realizar un buen diseño, conocer las principales características de cada una de ellas. A continuación se exponen los aspectos generales de los principales elementos de cada una de las etapas del sistema.

EL ENTORNO

El entorno está formado por un conjunto de informaciones sobre determinados elementos concretos de conocimiento. Se distinguen dos tipos de informaciones:

- La Tabla Objeto Atributo, tal como se definió en la sección 2, formada por un conjunto de ejemplos o elementos concretos de conocimiento definidos extensionalmente y por un conjunto de atributos asociados a dichos elementos, también representadas extensionalmente, de forma que todos los elementos vienen descritos extensionalmente en términos de los atributos.
- Un conjunto de *informaciones adicionales (I.A.)* sobre los elementos de la OAT y las características de la aproximación deseada.

Algunos aspectos importantes relacionados con estas componentes son:

LA TABLA OBJETO ATRIBUTO COMO MODELO

a) *La estructura del modelo*. La importancia de la representación del conocimiento inicial radica en dos aspectos fundamentales:

- Una representación adecuada de los conocimientos facilita la observación de sus características y el desarrollo de inferencias.
- El tipo de representación es un factor determinante en la eficiencia del sistema.

En [1] se utiliza la estructura *tabla* para representar el conjunto de conocimientos del modelo, con tantas filas como elementos concretos de conocimiento contenga el modelo y tantas columnas como atributos se utilizen para caracterizar dichos elementos.

b) El tipo de los atributos. Constituye otro factor decisivo en cuanto a la eficacia y eficiencia del sistema. Algunas de sus principales influencias son:

- Influencia en la calidad de las decisiones que generará el sistema. La calidad de la descripción depende de la descripción de los atributos. Por ejemplo, decisiones tomadas por inspección de atributos con valores aproximados serán, a primera vista, de menor calidad que las decisiones tomadas por inspección de atributos con valores precisos.
- Influencia en el rendimiento del sistema. La cantidad de recursos consumidos para llevar a cabo la toma de decisiones, en especial al consumo de tiempo y memoria depende del tipo de los atributos. Por ejemplo, trabajar con atributos cuyo dominio de valores es un subconjunto de los números reales, resulta en general más costoso que trabajar con atributos cuyo dominio de valores es un subconjunto de los enteros, puesto que las dificultades que supone el tratamiento de la precisión exigida por los reales exigen por lo general procedimientos adicionales.

c) La cantidad de elementos. La cantidad de elementos concretos de conocimiento y atributos del modelo es otro de los principales factores determinantes de la eficacia y eficiencia del sistema.

d) La calidad del modelo. El análisis de la calidad de la información del modelo constituye un importante aspecto en cuanto a la calidad de la descripción de los subconjuntos. En el caso que nos ocupa, la calidad de la información del modelo se refiere a la precisión con que han sido descritos los ejemplos del modelo mediante los atributos.

LAS INFORMACIONES ADICIONALES (I.A)

Forman un conjunto de informaciones sobre los elementos de la OAT que el descriptor precisa para la obtención de una descripción óptima. Se ditinguen de forma general los siguientes tipos de I.A.:

i. Informaciones relativas al coste de los atributos. Se considera que la determinación del valor que adoptan los ejemplos respecto a los atributos tiene un coste asociado, el cual puede diferir de un atributo a otro, pudiendo interpretarse de diferentes formas según las características del problema, tales como el peso de los atributos, valor de los atributos, ...

ii. Informaciones referentes al tratamiento de los atributos. Indican la forma en que deben tratarse los atributos para obtener una descripción de los subconjuntos que más se ajuste a las características del problema. De esta forma se han establecido diferentes tipos de tratamiento según el tipo de los atributos (distinguimos entre atributos binarios y

multivaluados) y según la posibilidad de que no existan bases de atributos en la OAT para poder describir los subconjuntos.

iii. Informaciones referentes a la descripción óptima. Se trata de especificar las características de la descripción óptima que desea obtenerse. Dichas características se exponen en la etapa correspondiente a la descripción óptima.

iv. Informaciones que especifiquen cuáles son los subconjuntos que desean describirse intensionalmente mediante los atributos de la OAT y cuáles sus características. La SDT definida en la sección 2 forma parte de esta clase de informaciones adicionales.

LA BASE DE CONOCIMIENTOS

La última etapa y objetivo del sistema consiste en obtener una descripción intensional óptima de los subconjuntos definidos extensionalmente en la SDT, que se almacena en la base de conocimientos. A efectos de este trabajo, identificaremos el término base de conocimientos con el de descripción óptima. La presentación de esta etapa anteriormente a la segunda se realiza exclusivamente por motivos de claridad de exposición.

Algunos aspectos importantes que surgen a la hora de establecer y aplicar el criterio que conduzca a la descripción óptima deseada son:

1) *Atributos que participarán en la descripción.* Hay que precisar las características de los atributos que se usarán en la descripción. En este punto deben tenerse en cuenta aspectos tales como el tipo y coste de los atributos.

2) *Características de la expresión asociada a la descripción.* Algunas de las principales características de la expresión que soportará la descripción de los subconjuntos son las siguientes:

- El tipo de estructura soporte de la expresión. La importancia del tipo de estructura viene dada por el tipo de aplicación, es decir, según la finalidad del uso de la descripción interesará más un tipo de estructura que otro. Por ejemplo, la estructura *árbol de decisión* resulta apropiada en caso de una aplicación interactiva de los conocimientos adquiridos, mientras que un conjunto de *reglas* es más idóneo para el desarrollo de inferencias.
- Características requeridas por el criterio a la estructura concreta utilizada. Se refiere a los casos en los cuales se exige que las relaciones definidas por la estructura satisfazcan determinadas propiedades. En estas situaciones se deben traducir las características de la descripción óptima en términos de las relaciones estructurales que deben satisfacer las expresiones óptimas.

3) *Obtención de una descripción óptima en caso de no existencia de bases.* En caso de que la OAT no contenga ningún subconjunto de atributos que definan sin ningúna de confusión los subconjuntos de la SDT, se dice que el modelo no posee bases. Esto ocurre cuando existe al

menos una pareja de ejemplos del modelo que poseen idénticas n-eplas de valores respecto a los atributos condición, pero difierentes w-eplas de valores respecto a los atributos consecuencia. En este caso, una buena idea consiste, por una parte, en clasificar los ejemplos de la OAT, de forma que cada clase consiste de todos los ejemplos con idéntica descripción respecto a los atributos condición (es decir, con idénticas n-eplas) y por otra en reescribir la información que proporcionan los atributos de la SDT con objeto de especificar de alguna forma la relación de cada clase con las w-eplas de valores de los atributos de la SDT.

EL DISPOSITIVO DE APRENDIZAJE

El dispositivo de aprendizaje constituye la segunda etapa y su función consiste en obtener una descripción óptima de los subconjuntos especificados en la SDT a partir de las características de la primera y tercera etapas. Para ello se basa en los dos siguientes pasos bien definidos:

1. Encontrar un subconjunto óptimo de atributos condición, R_b, $R \supseteq R_b$, que serán utilizados para describir los atributos consecuencia de R'. Se trata de encontrar una base de atributos condición R_b, $R \supseteq R_b$, respecto a los atributos consecuencia de R' que sea óptima.
2. Obtener una descripción intensional óptima de cada uno de los atributos consecuencia expresada en términos de los atributos de la base R_b.

Los criterios de optimalidad asociados a cada uno de los pasos son completamente independientes: una vez obtenida una base óptima, la descripción intensional se obtiene por aplicación del correspondiente criterio de optimalidad a los atributos de la base, sin necesidad de recomputarla de nuevo. Además, ambos criterios dependen exclusivamente de la naturaleza del problema.

El dispositivo de aprendizaje es por tanto una etapa procedural, cuyos aspectos dependen de las anteriores etapas. Se denomina *descriptor* al dispositivo de aprendizaje, por entender que es el término que mejor se ajusta a las funciones que tiene asignadas. Otras importantes funciones son:

- El descriptor debe tomar las consecuentes medidas derivadas del tipo de los atributos de la OAT.
- Se encarga del tratamiento de la OAT en caso de no existir bases.
- Elegir el método de resolución adecuado según las características del problema.

En la figura 3 se ilustran gráfiamente las etapas que componen el diseño de un sistema de adquisición inductia de conocimiento, de acuerdo con los términos utilizados.

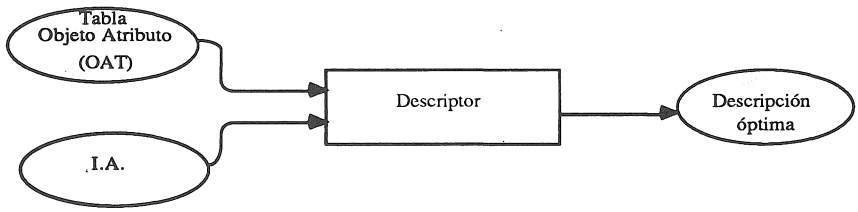


Figura 3. Modelo general de un sistema de adquisición inductiva de conocimiento.

4. CONCLUSIONES

La problemática de la adquisición inductiva de conocimiento se apoya, a nuestro modo de ver, en dos aspectos fundamentales:

- En el establecimiento de métodos de generalización de la información descrita extensionalmente mediante unos elementos concretos de conocimiento u ejemplos.
- En la mecanización del proceso diseño, surgido de la necesidad de establecer metodologías de que permitan abordar la solución de cualquier problema relacionado con el tema.

En cuanto al primer aspecto, son numerosos los métodos establecidos. No ocurre lo mismo respecto al segundo aspecto, debido esencialmente a la disparidad de métodos de generalización concebidos.

Proponemos el diseño de cualquier tipo de sistema de adquisición inductiva de conocimiento a partir de ejemplos, formado por tres etapas fundamentales:

- Primera etapa: La Tabla Objeto Atributo (OAT) y las Informaciones Adicionales. En esta etapa deben especificarse las características del modelo especificador de los ejemplos junto con la información referente a las características y tratamiento de los elementos del modelo y de la descripción óptima.

- Segunda etapa: El Dispositivo de Aprendizaje. Es una etapa procedural, enargada de otener la descripción óptima a partir de las características de la primera y tercera etapas. Para ello se basa en dos pasos fundamentales:

    - Obtener, según un criterio de optimalidad ajustado al caso, un subconjunto óptimo de atributos condición de forma que constituyan una base de atributos.
    - Obtener, a partir de la base del punto anterior y según un criterio de optimalidad ajustado al caso, una descripción intensional de los atributos consecuencia.

El criterio de optimalidad aplicado en cada etapa dependerá exclusivamente de la naturaleza del problema, reflejada en las características de las etapas del sistema.

- Tercera etapa: La descripción óptima. Está formada por los conocimientos adquiridos de la OAT mediante el descriptor, adecuadamente representados.

ABSTRACTO

La adquisición inductiva de conocimiento es uno de los campos de la Inteligencia Artificial que más atención está recibiendo en la actualidad. En el presente trabajo se presenta una visión general de su problemática, que a nuestro juicio se apoya en dos aspectos fundamentales:

- En el establecimiento de una aproximación de la solución que sea óptima en algún sentido, lo cual se traduce en la especificación de una descripción intensional óptima de algunos subconjuntos determinados.
- En el establecimiento de un proceso de diseño de sistemas de este tipo que permita abordar la solución de cualquier problema relacionado con el tema.

Se ha observado que la solución a cada uno de los anteriores aspectos depende en gran medida de las características de las etapas del proceso de adquisición inductiva de conocimiento.

Key words: Aprendizaje de las máquinas, adquisición inductiva de conocimiento, tabla objeto atributo (OAT), tabla de definición de subconjuntos (SDT), base de atributos.

5. BIBLIOGRAFIA

[1] G. Fiol. *Contribución a la adquisición inductiva de conocimiento.* PhD thesis, Universitat de les Illes Balears, Palma de Mallorca, Spain, 1991.

[2] G. Fiol, J. Miró. A New Perspective in the Inductive Acquisition of Knowledge from Examples. Aceptado en el congreso *IPMU'92 (International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems).* Palma de Mallorca, 6-10 de julio de 1992.

[3] G. Fiol, J. Miró. Theoretical considerations about the Process of Inductive Acquisition of Knowledge from Examples. Sometido a revisión a *12th World Computer Congress IFIP Congress'92.* Madrid, 7-11 de septiembre de 1992.

[4] G. Fiol, J. Miró. S.A.I.C., un sistema de adquisición inductiva de conocimiento. *Revista de Ciència.* Volumen nº 7, 61-78. Diciembre, 1990.

[5] R. S. Michalski and J. B. Larson. Selection of most representative training examples and incremental generation of VL1 hypotheses: The underlying methodology and the description of programs ESEL and AQ11. Technical Repord 867, Computer Science Department. University of Illinois at Urbana-Champaign, Urbana, IL, 1978.

[6] J. Miró. On defining a set by a property. Technical Repor[illegible] niversitat de les Illes Balears, Palma de Mallorca, Spain, 1987.

[7] Z. Pawlak. Rough Sets. *International Journal on Info[illegible]ation and Computer Sciences*, 11:341-356, 1982.

[8] Z. Pawlak. On Superfluous Attributes in Knowledge r[illegible]esentation Systems. *Bulletin of the Polish Academy of Sciences*, 32(3-4), 1984.

[9] Z. Pawlak. Rough Sets and Fuzzy Sets. *Fuzzy Sets and System*, 17:99-102, 1985.

[10] J. R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81-106, 1986.

[11] C.E. Shanon. A mathematical theory of communication. *Bell System Technical Journal,* Vol. 4, pp 379-423. 1948.

APLICACION DE CLP PARA MANTENER LA RAZÓN EN UN SISTEMA DE CONTROL CON RESTRICCIONES

Forradellas, R. *[A],[B]*
Ibañez, F. *[A],[B]*
Toledo, F. *[A],[B]*
Berlanga, R. *[A]*
Barber, F. *[A]*
Martin, G. *[A]*

[A] LISITT- Universidad de Valencia
C/Dr.Moliner 50 -46100 Burjassot -Valencia- España
FAX: +34-6-386 4568
E-mail: lisitt@evalun11.bitnet
lisitt@vm.ci.uv.es

[B] Universitat Jaume I
Campus de Penyeta Roja-Castellón-España
FAX: +34-964-345842
E-mail: forradel@inf.uji.es

RESUMEN:

En este trabajo se presentan diferentes métodos basados en técnicas de CSP [6] que permiten, mediante un lenguaje CLP, mantener la consistencia en un Sistema de Control con Restricciones sobre el que se realiza la aplicación.

Las diferentes técnicas están implementadas a través de aproximaciones realizadas en el Lenguaje CHIP [5], [7], [14] (Constraints Handling In Prolog) sobre las que se discuten las ventajas y desventajas que surgen entre ellas y con respecto a los métodos convencionales utilizados para el mantenimiento de la razón.

Palabras Clave: Mantenimiento de la Razón, Sistemas de Control, CLP.

Este trabajo fue parcialmente financiado por el Proyecto Esprit 5291 "CHIC" (Constraint Handling in Industry and Commerce), de la CEE.

1.- INTRODUCCION

Los Sistemas Basados en el Conocimiento y Razonamiento (SBCR) [16] tienen como objetivo *resolver problemas* por medio de la interacción de sus dos componentes principales: el *conocimiento*, visto como el conjunto de conceptos y sus interrelaciones, y el *razonamiento*, que es quien opera sobre el conocimiento, posibilitando que nuevos conceptos sean derivados a partir de otros.

El razonamiento se realiza cuando es posible inferir una información a partir de datos iniciales conocidos. Una de las formas más general de razonamiento es el No Monotónico. En este caso los Sistemas requieren un tratamiento especial, con el objeto de evitar los riesgos que pueden ocasionar una elección no correcta de soluciones. En otras palabras, el camino seleccionado en cualquier punto del árbol de búsqueda puede no conducir a la solución, en cuyo caso es necesario retornar a un punto anterior y reiniciar la búsqueda por otro camino. Este tratamieno, lo utilizaremos para encontrar soluciones mediante diferentes mecanismos de control propios del lenguaje CLP utilizados en nuestra aplicación [5], [7], [14].

Trataremos un Sistema de Control con Restricciones (SCR) en el cual es necesario mantener la razón, a partir de un conjunto inicial de creencias. Estas están almacenadas en la Base del Conocimiento, y están compuestas por *Suposiciones* y *Restricciones*. Para nuestro caso, el problema general consiste en mantener la razón del Sistema completo. Para lograr este objetivo se considera que el Sistema es Consistente, *si y sólo si* cada una de las Restricciones es consistente con el conjunto de Suposiciones. Cuando el Sistema se torna Inconsistente se debe producir una modificación de la información con el objeto de eliminar la contradicción. Esta modificación puede dar lugar a una inconsistencia motivada por la acción de otra restricción, por lo que nuevamente se producirán modificaciones y así sucesivamente.

El SCR que analizaremos es un Sistema de Control de Tráfico Urbano (SCTU) [1], [2], [3], [4], en el cual trataremos el problema de un *cruce urbano*, en principio de complejidad exponencial, que es el principal factor donde se produce explosión combinatoria. Esto implica trabajar con un árbol de búsqueda computacionalmente intratable con los métodos tradicionales y es donde hay que extremar esfuerzos para tornar tratable el problema y encontrar soluciones suficientemente cercanas al comportameinto real del sistema.

1.1.- Mantenimiento de la Razón

Convencionalmente, [11] los Sistemas para el Mantenimientode la Razón (SMR) son sistemas que permiten determinar un conjunto actual de creencias, a partir del conjunto actual de razones. El razonamiento se realiza a partir de las proposiciones almacenadas en la Base de Conocimiento -BC-. Cuando son encontradas nuevas razones, se actualiza el conjunto de creencias

El proceso que realiza el SMR consiste en explorar alternativas, realizar elecciones, explorar las consecuencias de las elecciones y por último comparar los resultados obtenidos cuando se utilizan elecciones diferentes. Si durante este proceso, se detecta alguna contradicción, el SMR actualiza la BC y cambia sus creencias con el propósito de eliminar la contradicción.

En la mayoría de los modelos desarrollados [11], [17], [18], [19], el SMR interactúa con un Solucionador de Problemas (SP), el cual transfiere al SMR las inferencias que realiza, y el SMR utiliza la estructura de estas inferencias para organizar las creencias del SP. La forma de interacción entre los dos subsistemas

varía, pero normalmente pueden considerarse que el SMR enviará información al SP sobre las creencias en que se basan las inconsistencias y sobre los conjuntos de creencias mutuamente consistentes con los que se puede trabajar. Toda la inferencia que se haga es comunicada al SMR por el SP. Por otro lado los modelos analizados no contemplan esta inferencia (cómo obtener nuevas creencias a partir de las existentes), siendo el SP quien contempla tal problema.

1.2.- Programación Lógica con Restricciones -CLP-

Muchos problemas de la vida real pueden ser tratatos el mediante manejo de restricciones. Tradicionalmente, la aproximación más común para resolver estos problemas consistía en tratar los problemas con lenguajes procedurales. Estas aproximaciones requieren sustanciales esfuerzos por parte del programador para el desarrollo de programas, y el producto terminado resulta difícil de mantener, modificar y/o adaptar. La programación lógica resulta muy apropiada en términos de declarativiidad, para el tratamiento de problemas con restricciones. Sin embargo, los lenguajes de programación lógica del tipo Prolog en la práctica resultan ineficientes para atacar problemas con explosión combinatoria considerable.

La Programación Lógica con Restricciones (Constraints Logic Programming, CLP) está basada sobre la idea de sustituir la unificación, la operación básica de los lenguajes lógicos tradicionales, por el concepto más general de resolución de restricciones sobre un dominio de computación. Una de las ventajas de los lenguajes CLP, es la facilidad de programación, lo que permite al programador razonar directamente en términos del dominio, en lugar de forzar la codificación a la semántica de objetos en términos del Universo de Herbrand. Otra ventaja es la eficiencia, que resulta del uso de algoritmos especializados para la resolución de restricciones sobre el nuevo dominio de computación. De aquí, resulta que CLP combina la declarativiidad y la flexibilidad de la programación lógica, con la eficiencia de aproximaciones procedurales.

Una razón para el éxito de CLP en recientes aplicaciones es la elección de dominios de computación integrados dentro de diferentes sistemas. En los últimos años, se han definido diversos lenguajes CLP. Estos, básicamente difieren en los dominios de computación sobre los que trabajan y/o los algoritmos para resolución de restricciones que usan.

1.2.1.- El Lenguaje CLP CHIP

CHIP (Constraints Handling In Prolog) es un nuevo lenguaje CLP que combina los aspectos declarativos de la programación lógica con la eficiencia de las técnicas de resolucion de restricciones [10], [12], [13]; fue desarrollado en el ECRC (European Computer-Industry Research Centre, fundado por BULL, SIEMENS e ICL) con el fin de abordar problemas del mundo real con un gran árbol de búsqueda y está basado en el uso activo de restricciones. De la versión original han surgido los productos: *Charme* de BULL, *Decision Power* de ICL, *Pecos* de ILOG y *CHIP* de COSYTEC, que es la versión que actualmente utilizamos en el marco del Proyecto Esprit "CHIC" [14].

De acuerdo con los criterios presentados, CHIP incluye tres nuevos dominios computacionales [8]: *booleanos*, *racionales*, y *dominios finitos*. Para cada uno de ellos se utilizan, diferentes técnicas de resolución de restricciones especializadas:

resolución de ecuaciones en el álgebra booleana, algoritmos tipo simplex, y técnicas de consistencia, respectivamente.

En nuestra aproximación utilizaremos *dominios finitos*, cuya característica básica es la de trabajar sobre *variables dominio*, que son variables con un rango de valores comprendido en un conjunto finito. CHIP diferencia entre dos clases distintas de tales variables, unas comprendidas en un rango de valores constantes y otras con un rango de valores comprendido en un conjunto finito de los números naturales.

Una declaración de dominios [5] puede verse en el siguiente predicado:

```
P(X,Y,Z):-X::1..9, Y::[1,2,7],...
```

En esta construcción, la declaración de dominios se emplea como un subobjetivo Prolog que tiene éxito y las variables libres que se colocan a la izquierda del operador "::" toman como dominio el expresado en la parte derecha de este operador. El predicado **P** tiene como primeros subobjetivos, las declaraciones de dominio de las variables **X** e **Y**. Estas variables pasan a ser inmediatamente variables dominio y podrán efectuarse sobre ellas todas las operaciones de reducción que se definan posteriormente. La variable **X** toma valores enteros sobre el intervalo [1,9], la variable **Y** toma valores en el conjunto {1,2,7}. Mientras que la variable **Z** (sobre la que no se ha definido ningún dominio) tomará sus valores en el universo de Herbrand.

Todas las restricciones implicadas en variables dominio son resueltas a través de técnicas de consistencia [6], [10], [12], [13]. La principal característica de estas técnicas es la de reducir el tamaño del espacio de búsqueda mediante los pasos siguientes:

Paso 1: Propagar las restricciones tanto como sea posible.
Paso 2: Realizar elecciones.

Estas técnicas de consistencia [13] ofrecen diferentes maneras de realizar la poda del espacio de soluciones, y se basan en las reglas de inferencia: Forward Checking (FCIR), Looking Ahead (LAIR) y Partial Looking Ahead (PLAIR).

2.- UN SMR CON RESTRICCIONES ABORDADO CON TECNICAS CLP

El mecanismo del SMR que vamos a proponer está enfocado a sistemas que almacenan la información mediante variables numéricas en el dominio de los naturales. La BC del SCR sobre el que es necesario mantener la razón, tiene almacenada un conjunto inicial de creencias (conocimiento inicial), compuesto por *Suposiciones* y *Restricciones*.

·*Suposiciones*: se definen como un tipo particular de Fórmulas Bien Formadas (FBF) o Proposiciones, las que se construyen por medio de un conjunto finito de disjunciones, de la forma:

$$A_j)\ \text{eq}\ (X_j,v_1)\ \text{or}\ \text{eq}\ (X_j,v_2)\ \text{or}\ \ldots.\ \text{or}\ \text{eq}\ (X_j,v_n).$$

Donde,

A_j es la suposición j de la base de suposiciones,
X_j es la j-ésima variable que describe el Sistema,
v_i $(0<i<n)$ son los valores constantes asociados a la variable X_j,
eq/2 es un predicado que liga la variable X_j con la constante v_i.

·*Restricciones*: se construyen a partir de operaciones relacionales entre dos tipos de términos (lineales y funcionales), que serán definidos a continuación.

Previo a describir la forma que tomarán las restricciones, haremos las siguientes definiciones preliminares:

- Sean $X_1..X_m$ variables lógicas.
- Para cada variable Xi definimos su clausura del siguiente modo:

$$cl(X_i) = \{a \,/\, eq(X_i,a) \models A_i\}$$

- Sean $F_1,..,F_k$ funciones definidas sobre variables lógicas de la forma

$$F_j: cl(X_j) \rightarrow n \quad (1 \leq j \leq k)$$

- Término Lineal -TL- se define recursivamente del siguiente modo:
  - Un número natural es un TL.
  - Si *c* es un número natural y X_i es una variable, entonces $c \cdot X_i$ es un TL.
  - Si T1 y T2 son TL entonces T1+T2 es un TL.
- Término Funcional -TF- se define recursivamente del siguiente modo:
  - Un número natural es un TF.
  - Si Xi es una variable, c es un número natural y Fj es una función como las que hemos definido al inicio de la sección, entonces c·F(Xj) es un TF.
  - Si T1 y T2 son dos TF, entonces T1+T2 es un TF.

La forma que tendrán las restricciones es:

Restricciones Tipo1

$$<TL_1> <op_rel> <TL_2>$$

Donde: $<TL_1>$ y $<TL_2>$ son TL y <op_rel> es una de las relaciones: >=, >, <, <= o =

Veamos un ejemplo; sean las suposiciones:

A_1) $eq(X_1,1)$ or $eq(X_1,2)$

A_2) $eq(X_2,3)$ or $eq(X_2,4)$ or $eq(X_2,10)$

A_3) $eq(X_3,5)$ or $eq(X_3,6)$

A_4) $eq(X_4,7)$ or $eq(X_4,6)$ or $eq(X_4,7)$

Ejemplo: una Restricción de Tipo 1 sería: $2X_1 + 3X_4 = X_2 + 2X_3$

Restricciones Tipo 2

$$<TF_1> <op_rel> <TF_2>$$

Donde: $<TF_1>$ y $<TF_2>$ son TF y <op_rel> ídem al anterior.

Por ejemplo: sean las suposiciones anteriores $A_1..A_4$, entonces una restricción posible sería: $2F_1(X_1) + 3F_2(X_2) = 5F_3(X_3) + 4F_4(X_4)$

Diremos que un Sistema de esta clase es consistente si y sólo si cada una de las restricciones, es consistente con el conjunto de proposiciones A_i. Para cada una de las restricciones descritas tendremos las siguientes situaciones:

Una restricción de Tipo 1 es consistente respecto al conjunto de Suposiciones si y sólo si Para todo i, $1 < i < k$ y para todo *a* , tal que $eq(X_i,a) \models A_i$ existen $v_1,...,v_{i-1},v_{i+1},...,v_m$ tal que,

$eq(X_1,v_1)$ and ...and $eq(X_{i-1},v_{i-1})$ and $eq(X_{i+1},v_{i+1})$ and ...and $eq(X_k,v_k)$
$\models A_1$ and A_2 and ... and A_{i-1} and A_{i+1} and ... and A_k.

y verifiquen la restricción dada, sustituyendo las variables

X_j, por los valores v_j $(1<j<m$ y $j=i)$
y X_i por *a*.

Para una restricción de Tipo 2 la consistencia respecto del conjunto de suposiciones se describe exactamente igual que para las restricciones de Tipo 1, excepto que se deben evaluar las funciones sobre las variables.

Cuando el SMR se torna inconsistente por algún motivo, debe producirse una modificación de la información de manera que hagamos desaparecer la contradicción aparecida. Esta modificación consiste en la eliminación de predicados eq/2 de las disjunciones que produzcan inconsistencia, es decir que no cumplan las condiciones de consistencia antes expresadas, a su vez esta modificación puede hacer inconsistente la hipótesis con alguna restricción produciendo una propagación de modificaciones sobre las hipótesis hasta que el sistema completo se torne consistente.

Si una restricción produce la eliminación de todos los predicados eq/2 de la misma suposición, entonces la restricción es incompatible con las suposiciones y deberá ser eliminada.

4.- APLICACIÓN

4.1.- El problema del Tráfico Urbano

El mecanismo actual para la gestión del tráfico en una red urbana [15] consiste en la asignación de tiempos de verde y rojo en cada una de las intersecciones por medio de *semáforos*. Un semáforo asigna de forma alternativa el derecho de paso a cada movimiento o grupo de movimientos que confluyen en una intersección. Esta asignación se realiza fijando dos valores: *Ciclo*, que es el número de segundos en que un semáforo en una intersección tarda en repetir su comportamiento y *Reparto*: que es el porcentaje de verde y rojo de cada uno de los movimientos posibles en una intersección para un determinado ciclo.

Mientras que el reparto está asociado a una intersección, el ciclo está asociado a un conjunto de intersecciones que se denominan Controladores de Ciclo Idéntico (CCI) que se encargan de regular el tráfico en una zona, de esta forma la red está dividida en zonas (CCI) y cada zona está compuesta por varias calles e intersecciones. De esta manera la gestión de tráfico en una ciudad puede ser reducida al manejo de un número de zonas de CCI. Para una eficiente gestión, el SCTU debe efectuar un análisis permanente del estado del tráfico, mediante detectores repartidos en la red, que permitirá la toma de decisiones en un sistema centralizado, él que transmitirá a los elementos reguladores de tráfico (semáforos), las soluciones encontradas.

El SCTU que tratamos incluye un simulador cuya función es la de proveer información de vital impotancia para la toma de decisiones. En este trabajo trataremos el problema que se presenta en una intersección con *m* entradas y *n* salidas, que es el principal factor donde se produce una explosión combinatoria sobre la cual realizaremos aproximaciones para el mantenimiento de la razón.

4.1.1.- Restricciones

En el Sistema presentado, el estado del mismo se describe mediante variables, que representan las densidades de vehículos, correspondientes a las entradas y salidas en una intersección, las cuales pueden tomar un número fijo de valores posibles. Estas variables (representadas en CHIP mediante variables dominio) están sometidas a verificar restricciones sobre densidades. Además a cada densidad le corresponde un flujo y por lo tanto, éstas variables tienen asociadas funciones con codominio real para representar el flujo. Mediante estas funciones se construyen TF, los cuales también deben verificar restricciones.

Cada vez que las variables dominio: $x_1, ..., x_{m+n}$ (para m entradas y n salidas) cambian sus valores, se deben satisfacer las siguientes restricciones:

- **De Transición** (RT): que dependen de los valores iniciales $x_1, ..., x_{m+n}$.
- **Instantáneas** (RI): que dependen sólo del valor de la variable_dominio asociada al al reparto (V), que toma valores en el conjunto F=$\{Fase_1, ..., Fase_k\}$ para k repartos.

Las RT restringen los posibles valores de las variables que representan las entradas y las salidas. Esta reducción se efectúa sólo una vez sin producir propagación [2], por lo tanto no es necesaria ninguna técnica de consistencia sofisticada para tratarlas. Las RI han sido formalizadas en anteriores presentaciones [2], [3], [4], [14] y se ha demostrado que para una determinada fase se representan mediante las fórmulas:

$$F(x_i) \doteq \sum_{j=1}^{m} \alpha_{ij} \cdot F(x_j) \qquad \text{para } m+1 \leq i \leq m+n$$

$$F(x_i) \doteq [0,0] \text{ para todas las entradas donde los vehículos no circulan.}$$

Estas fórmulas representan el flujo de las salidas en función del flujo de las entradas y serán representadas como Restricciones Tipo 2.

Donde: $F(x_i)$ es el intervalo $[F_{min}(x), F_{max}(x)]$, con F_{min} representando el mínimo flujo asociado a x, y F_{max} el máximo,
y la relación $\doteq$ entre 2 intervalos se define: $I_1 \doteq I_2$ Sii, $I_1 \cap I_2 = \varnothing$

4.2.- Aproximaciones del problema con CHIP

Las hipótesis pueden ser representadas en CHIP mediante *variables dominio*. Definiremos una variable por cada hipótesis, siendo su dominio, los posibles valores que tenía asociada la variable a partir de los predicados eq/2 que constituían dicha hipótesis, definido anteriormente como clausura de una variable. Para el tratamiento de las restricciones de Tipo 2 se utilizarán las reglas de inferencia FCIR y PLAIR cuyos mecanismos de control se describen brevemente:

4.2.1.- Forward Checking -FC-

Este es un procedimiento de búsqueda eficiente, especializado para Constraints Satisfaction Problems (CSPs) [9] que consiste en reducir el conjunto finito de valores posibles que puedan ser asignados a una variable. De esta manera se poda el espacio

de búsqueda *a priori* para eliminar la combinación de valores que no pueden dar lugar a una solución del problema.

Esta regla de inferencia permite el mecanismo de propagación de las restricciones, las que tienen la propiedad de que permanecen en espera hasta tener una sola variable_dominio no instanciada, estando el resto de las variables que forman parte de la restricción, instanciadas. Con ésta metodología podemos resolver eficientemente el problema planteado y veremos como se producen las reducciones sobre las variables dominio asociadas a las RI. Estas se basan en las condiciones que deben satisfacerse para que la intersección de dos intervalos sea no nula [4]. Estos intervalos están configurados de la siguiente manera: el primero corresponde al intervalo de flujo asociado a la densidad de una salida, y el segundo corresponde a la suma de los intervalos asociados a las entradas, los cuales están multiplicados cada uno ellos, por el coeficiente que asocia cada entrada con cada salida. Estos coeficientes forman una matriz que será distinta para cada fase

Dados los intervalos I_1 e I_2, $I_1 \cap I_2 = \varnothing$ Sii

[A] $F^-(I_2) > F^+(I_1)$ o [B] $F^+(I_2) < F^-(I_1)$

donde $F^-(I)$ (respectivamente $F^+(I)$) representa el extremo inferior (superior) del intervalo I. Así si [A] se cumple podemos asegurar que la intersección es vacía. El intervalo asociado a las entradas, es la suma de los intervalos asociados a cada entrada multiplicado por los correspondientes coeficientes, por lo tanto, suponiendo que I_1 representa una salida, si I_2 representa una sola entrada y la condición [A] se cumple, podemos asegurar que seguirá siendo cierta si I_2 representa todas las entradas, siendo posible, de esta forma, "podar" el espacio de búsqueda "a priori". Es por ello que podemos aplicar a cada pareja entrada-salida la condición [A] y descartar las densidades que lo cumplan, ya que es seguro que no producirán intersección. De esta manera podemos generar un conjunto de restricciones binarias que asociarán cada salida con cada entrada

4.2.2.- Partial Looking Ahead -PLA-

Looking Ahead (LA) es otro procedimiento de búsqueda utilizado en los problemas de combinatoria cuyo método difiere en el modo en que las restricciones pueden ser usadas; para el caso del FC sólo una variable debe estar desinstanciada, mientras que en LA pueden haber varias variables desinstanciadas. Para este caso el posible conjunto de variables es reducido, pero las restricciones no son totalmente satisfechas y pueden ser reconsideradas más tarde [12].

En otras palabras, el camino seleccionado en cualquier punto del árbol de búsqueda puede no conducir a la solución, en cuyo caso es posible retornar al punto original y encaminar la búsqueda por otro camino. Esta modificación puede dar lugar a una Inconsistencia motivada por la acción de otra restricción, por lo que nuevamente se producirán modificaciones y así sucesivamente hasta que el Sistema completo se torne Consistente.

El mecanismo de control PLA provee una base formal para la implentación de algunas restricciones. Esta implementación no es un caso particular de FC ni de LA, pero es posible considerarla como un caso intermedio entre ellos. La regla de inferencia PLAIR [5], [6] consiste en reemplazar algunos puntos, en la definición de LAIR utilizada en LA, para lo cual el conjunto de valores no son definidos en la regla

de inferencia y son dependientes de cada restricción en particular. Para este caso las restricciones tienen la propiedad de reducir el dominio de las variables, para que satisfagan la restricción. En esta aproximación utilizaremos el predicado "*element*" preuefinido en CHIP, para resolver el problema [5], [14].

- Supongamos una posible ecuación para las RI: $F(x_3) \doteq 0 \cdot F(x_1) + 0.5 \cdot F(x_2)$
  y las variables dominio: $_OFmin_3$, $_OFmax_3$, $_IFmin_1$, $_IFmax_1$, $_IFmin_2$, $_IFmax_2$
  para representar, $Fmin(x_3)$, $Fmax(x_3)$, $Fmin(x_1)$, $Fmax(x_1)$, $Fmin(x_2)$, $Fmax(x_2)$
- Obtenemos: $[_OFmin_1, _OFmax_1] = [0*_IFmin_1+0.5*_IFmin_2, 0*_IFmax_1+0.5*_IFmax_2]$

Considerando a *F(x)* y la relación entre 2 intervalos definidos en 4.1.1 queda:

$_OFmax_1 \geq 0*_IFmin_1 + 0.5*_IFmin_2$ and $_OFmin_1 \leq 0*_IFmax_1 + 0.5*_IFmax_2$ [*]

La solución se alcanza usando un par de predicados *element* para cada variable, uno para establecer la correspondencia con el flujo mínimo y otro para el flujo máximo. Por ejemplo, para x_1:

a) element(x_1,[0,90,265,...],$_IFmin_1$) y **b)** element(x_1,[90,265,375,...],$_IFmax_1$)

donde la lista [0,90,265,...], representa [$Fmin(x_1)$, $Fmin(x_2)$, $Fmin(x_3)$, ...],
y la lista [90,265,375,...], representa [$Fmax(x_1)$, $Fmax(x_2)$, $Fmax(x_3)$, ...]

multiplicados por el valor 100, puesto que CHIP trabaja sólo con naturales, las restricciones [*] se reescriben en CHIP de la siguiente forma:

100 * _OFmax1 #>= 0 * _IFmin1 + 50 * _IFmin2

Ahora, las reducciones se hacen sobre variables dominio que representan flujos. Si $_IFmin_1$ es reducida, entonces, x_1 será reducida como concecuencia del predicado *element* **a)**, pero esta redución despertará el predicado *element* **b)**, y consecuentemente será reducida la variable $_IFmax_1$. Ahora se despertarán todas las restricciones donde aparece $_IFmax_1$. y así sucesivamente, hasta tornar el sistema Consistente.

5.- CONCLUSIONES

En la aproximación implementada con FC, el tamaño del árbol de búsqueda es reducido, aunque provoca un gasto de tiempo adicional en cada nodo, la cantidad de nodos considerados se reduce y de aquí el número de restricciones manipuladas. De este manera, mediante la poda "a priori" antes del descubrimiento del fallo, se logra conseguir una notable ventaja frente al método de Bactraking Standard, Pero hay partes del árbol que no verifican las RI y sin embargo no son podadas, puesto que sólo estamos usando una condición suficiente.

En cuanto a la aproximación implementada con PLA el árbol es completamente "podado", puesto que usamos una condición necesaria y suficiente. De esta manera todas las soluciones generadas verifican las RI y viceversa. Con el uso de esta aproximación se logra además una forma más natural de resolver el problema.

Se logra un considerable ahorro de tiempo, pues si consideramos que el árbol inicial de búsqueda tiene 8^{n+m} nodos (n entradas y m salidas), al efectuar una "poda" inicial sobre las variables dominio, mediante las Restricciones del Sistema, se reducen los dominios de dichas variables y en consecuencia el número de combinaciones

posibles. El árbol inicial queda sustancialmente reducido. Por último, aunque el coste adicional en cada nodo del árbol es mayor, para un gran árbol de búsqueda la eficiencia de las restricciones supera ampliamente el coste adicional mencionado.

En el SMR presentado se obtienen resultados que presentan ventajas frente a la forma de abordar el problema utilizando metodologías tradicionales. Por un lado la metodología presentada, permite representar el problema en forma sistemática y declarativa con CLP. Por otro lado, se logra un considerable ahorro de tiempo de ejecución debido a dos razones: 1ra. PLAIR permite trabajar en un espacio de búsqueda sustancialmente reducido y 2da. el trabajo realizado en cada nodo, es obtenido con mínimo coste computacional debido al mecanismo asociado a PLAIR.

6.- AGRADECIMIENTOS

Parte de la investigación realizada en el presente trabajo ha sido realizada mientras los autores trabajaban en el Proyecto Esprit 5291 de la CEE "CHIC" [14] Organizaciones colaboradoras son: ECRC y SIEMENS (Alemania), ICL e Imperial College (Gran Bretaña), BULL, Reanault, CERT/ONERA y Dassault (Francia), AIS y Braghenti (Italia), CMSU (Grecia) y OCT (España).

7.- REFERENCIAS

[1] Martín G., Toledo F. & Moreno S. "KBS for Road Network Traffic Control State of the Art and Perspectives", AI & ES, Avignon '92.
[2] Ibañez F., Forradellas R., Toledo F., Martín G. y Moreno S. "Aplicación de CLP a la Simulación Cualitativa de la Gestión de Tráfico Urbano", AEPIA '91.
[3] Martín G., Toledo F., Ibañez F., y Forradellas R. "User Requirements Report for Urban Traffic Control in CHIP", CHIC Project, Technical Report, Munich 11/91.
[4] Ibañez F. Forradellas R. Berlanga, R. Toledo F. y Martín G. "Propagación de Constraints sobre el Tratamiento de un Cruce Urbano" Jornadas ProDe91.
[5] CHIP Reference Manual, ICL/COSYTEC 1991/92
[6] Van Hentenryck, P "Constraints Satisfaction in Logic Programming", MIT 1989.
[7] Forradellas R., Ibañez F., Berlanga R., Martín G. y Toledo F. "Manejo de Constraints en CHIP", Jornadas ProDe91.
[8] Van Hentenryck P., Dincbas M., "Domains in Logic Programming", IAAA 86
[9] Van Hentenryck P., Dincbas M., "Forward Checking in Logic Programming", FICLP, 5/87
[10] Dincbas M., Van Hentenryck P., Simonis H., Aggoun A., Graf T. and Berthier F "The Constraints Satisfaction Logic Programming Languaje CHIP", FGCS 88
[11] Botti V."Mantenimiento de la Razón en Sistemas Basados en el Conocimiento Dependientes del Tiempo", Tesis Doctoral, UPV, 1990
[12] Van Hentenryck P. "A Theorical Framework for Consistency Techniques in Logic Programming", IJCAI 87.
[13] Dincbas M., Van Hentenryck P., Simonis, "Extending Equation Solving and Constraint Handling In logic Programming", CREAS 87.
[14] Esprit "CHIC" Project, Second Review, Paris 1/92
[15] "Traffic Engineering Handbook", Prentice-Hall, 1989
[16] Rich, Elaine, "Artificial Intelligence", McGraw-Hill 1989
[17] Doyle J. "A Truth Maintenance System" Artificial Intelligence, 1979.
[18] De Kleer J., "An Assumtion-based TMS", Artificial Intelligence, 1986.
[19] McDermontt D., "Data Dependencies on Inequalities", AAAI-83.

Hacia la generación de reconocedores LR(1) de reducidas dimensiones

Fortes Gálvez, José
Departamento de Informática
Universidad de Las Palmas de Gran Canaria
Islas Canarias (España)

Abstract

Los métodos clásicos LR(1) presentan el problema de producir tablas de reconocimiento demasiado grandes para las gramáticas de lenguajes de programación. Una alternativa es construir un autómata que combine el símbolo de prelectura con la lectura de la pila del análisis sintáctico desde su cima, para determinar la siguiente acción de reconocimiento a realizar.

Aquí se exponen, junto a los desarrollos teóricos que han conducido a un método para construir este tipo de reconocedores, una breve descripción del mismo y de los resultados experimentales, siendo el aspecto más destacable las importantes reducciones que se han observado en tamaños de autómatas frente a los métodos estándar.

Palabras clave: análisis sintáctico, gramáticas LR(1), generadores, autómata inverso, procesadores de lenguajes

1 Introducción

Los métodos clásicos [11, 1, 17] de construcción de analizadores sintácticos para la clase de gramáticas LR(1) presentan el problema de producir autómatas o tablas de reconocimiento de enormes dimensiones en comparación con los tamaños de las gramáticas. Ello es especialmente grave para el caso de gramáticas de lenguajes de programación. Ante este problema, la solución que se ha adoptado ha sido la de utilizar reconocedores LALR(1), aún a costa de una pérdida en la capacidad de reconocimiento.

En los métodos de precedencia [15, 18, 9, 4] se utiliza en principio el par de símbolos (cima de la pila, ventana de prelectura) para decidir si se debe avanzar—almacenar el símbolo de la ventana en la cima—o reducir. El problema de lo limitado de la aplicabilidad de estos métodos radica en que es muy frecuente encontrar gramáticas en las que, para algunos casos, es necesario conocer algo más del contenido de la pila, lo que ha dado lugar a soluciones algo más potentes [5, 7, 12],

pero aún muy lejos de LR(1). A esto se suele añadir una limitación adicional en cuanto a las partes derechas de la gramática, y en particulár las reglas nulas están siempre prohibidas.

El enfoque que aquí se toma parte de la misma noción anterior, ampliamente experimentada como válida dado el gran número de métodos desarrollados a lo largo de muchos años, pero generalizándola hasta sus últimas consecuencias. Así, el método admite que, cuando sea necesario la pila se explore en la profundidad precisa, de forma que siempre se alcance una solución, por lo que el método garantiza la potencia LR(1). No se aplica ninguna restricción a las reglas de la gramática, y el analizador no sólo habrá de discriminar entre avance o reducción sino que, en este último caso, indicará qué regla aplicar.

En este artículo se introduce brevemente este novedoso método [6], que presenta las características antes citadas, junto con el desarrollo previo que nos ha llevado al mismo, para la generación de reconocedores LR(1) de pequeño tamaño.

2 Planteamiento del problema

Justo antes de la aparición de los métodos LR, el método que mejor compromiso [1] aportaba entre sencillez y eficiencia, por un lado, frente a potencia de reconocimiento, por otro, era el método de precedencia débil.

Su concepto base es muy claro: relación entre el símbolo superior de la pila y el de prelectura, con incidencia directa sobre el proceso de análisis ascendente, de forma que la relación $\lessdot$ implica avance, y la relación $\gtrdot$ implica reducción. El cálculo de tales relaciones es relativamente sencillo, y su implementación eficiente. Además, se conocen buenos métodos para incorporar el tratamiento automático de errores [8] al proceso de análisis. Incluso los posibles problemas de conflictos entre relaciones pueden ser solventados, aunque algo inelegantemente, mediante la introducción de nuevos símbolos no terminales.

Sin embargo, la precedencia débil presenta dos problemas que los métodos LR resolvieron de forma muy elegante. El primero de ellos la posibilidad de disponer de reglas nulas en la gramática, cosa que hasta entonces no se entendía muy bien, sobre todo desde el punto de vista de su utilidad práctica; concepto que hoy está claramente resuelto. Y el segundo, la necesidad práctica de incorporar algún mecanismo adicional para resolver los conflictos reducción-reducción, si no queremos estar realizando continuas transformaciones de reglas en la gramática. Es, fundamentalmente, con el objetivo de abordar este segundo problema, con el que surgen los métodos de exploración del contexto izquierdo [5, 7, 12].

Pero el impacto de los métodos LR es demasiado fuerte, y prácticamente hace desaparecer cualquier esperanza de que, trabajando en la mejora de los métodos de precedencia, pueda algún día alcanzarse algo comparable. Esto es especialmente cierto al aparecer el método LALR [3]. Posiblemente ha influido también su incorporación en forma de la utilidad `yacc` al sistema operativo Unix, y con ello su popularización y su conversión por tanto en método estándar de facto.

3 Mejorando los métodos de precedencia

Vamos, a pesar de todo, a plantearnos la posibilidad, aunque sólo sea circunscribiéndonos inicialmente a la exploración teórica, sin relevancia inmediata en el campo de su aplicación práctica, de la mejora de los métodos de precedencia, y en particular, de la precedencia débil. Así, de las dos desventajas que hemos mencionado para la precedencia débil, y que por tanto son los candidatos inmediatos a un estudio teórico para su mejora, podemos hacer unas primeras reflexiones.

En el caso de las reglas nulas, podemos decir que no son intrínsecas a la idea misma de precedencia, sino que más bien son consecuencia de la moda, por decirlo de alguna manera, imperante en el momento en que se desarrollan estos métodos. Así, cabe perfectamente replantearse la existencia de relaciones de precedencia débil, por ejemplo, que impliquen avance o reducción, considerando la posible existencia de reglas nulas.

Y en el caso de la exploración del contexto izquierdo para la resolución de conflictos reducción-reducción, podemos decir que se trata, simplemente, de que la precedencia débil pretende fundamentalmente permitirnos discernir entre cuándo se ha de avanzar, y cuándo reducir. Por tanto, este mecanismo está abierto a mejoras para la resolución de los citados conflictos.

Veamos, por tanto, cada uno de estos aspectos por separado.

3.1 Incorporación de reglas nulas

Corrobora nuestras afirmaciones anteriores constatar que solamente desde época muy reciente, ha recibido atención el estudio de la incorporación de reglas nulas a los métodos de precedencia, en particular [14, 13]. Aquí expondremos brevemente un método distinto, aunque con los mismos resultados finales, desarrollado independientemente por el autor.

A diferencia de [14], donde se desarrolla un nuevo mecanismo de cálculo de unas nuevas relaciones de precedencia, adaptadas a gramáticas con reglas nulas, nuestro interés es mantener una compatibilidad en los mecanismos de cálculo tradicionales, con las correcciones oportunas para contemplar las reglas nulas. Así, introducimos el concepto de *relación de regla nula*, de forma que todo símbolo no terminal que produce la secuencia nula ε introduce la correspondiente relación de precedencia. Más concretamente, decimos que dos símbolos $X \in V$, $c \in T$ están *N-relacionados* si $N \rightarrow \varepsilon$ y existe una forma sentencial canónica en la que X está en la cima de la pila, c es el símbolo de prelectura, y el handle ε ha de ser reducido a N. De esta forma, suponiendo que no haya conflicto, el reconocedor puede decidir si avanzar ($\lessdot$), reducir ε (N-relación) o reducir una regla no nula ($\gtrdot$).

Para el cálculo de las relaciones de regla nula introducimos la noción de gramática con terminales de regla nula, donde cada regla nula $N \rightarrow \varepsilon$ es sustituida por $N \rightarrow \mathcal{E}_N$, donde $\mathcal{E}_N$ es ahora considerado un símbolo terminal, de forma que la gramática queda convertida a gramática sin reglas nulas, para conservar el esquema tradicional de cálculo, como era nuestro objetivo inicial. En este mismo sentido,

es destacable que las nuevas relaciones son calculables a partir de las tradicionales relaciones de precedencia aplicadas a la gramática que incorpora los terminales de regla nula.

De la misma manera que con el caso tradicional, es posible realizar transformaciones automáticas mediante la introducción de nuevos símbolos no terminales, de forma que desaparezcan los conflictos de precedencia—en realidad quedan convertidos en problemas de contexto izquierdo, que es el tema que pasamos a estudiar a continuación. En cualquier caso, es destacable que la aplicabilidad práctica de la precedencia débil aumenta en sí misma notablemente con la incorporación de reglas nulas, siempre que estemos dispuestos a asumir la algo fastidiosa transformación de reglas.

3.2 Resolución de conflictos reducción-reducción

Es necesario abordar la resolución de conflictos reducción- reducción para una verdadera utilidad práctica del método, en comparación a los conocidos actualmente. Ya hemos indicado que en su momento se desarrollaron métodos que hacen uso de un número limitado- -y necesariamente pequeño por condicionantes propios del diseño—de los símbolos en la cima de la pila, para este propósito. Igualmente, sólo recientemente ha recibido de nuevo atención esta línea de trabajo, en particular en el tratamiento de errores [16, 2].

Sin embargo, está claro que por esta vía será difícil obtener algo similar a la potencia de, por ejemplo, LR(1) por la sencilla razón de que, por definición, el método LR(1) utiliza toda la información disponible en la pila más el símbolo de prelectura para, no sólo decidir entre avanzar o reducir, sino además saber qué regla es la que debe utilizarse en la eventual reducción.

Planteémosnos pues, aunque sea en principio como ejercicio puramente teórico, la posibilidad—necesaria por otro lado por lo que acabamos de ver—de utilizar *toda* la información de la pila para decidir en conflictos reducción-reducción. Es fácil ver que, en este caso, y combinando lo visto en la sección precedente, nos encontraríamos con potencias de análisis sintáctico próximas (aunque quizá algo inferiores) a LR(1). Evidentemente, el posible resultado merece el esfuerzo de intentarlo.

La solución desarrollada por el autor es la de construir un autómata finito determinista que, explorando la pila desde su cima hacia la base, en el momento de aplicar una reducción, sea utilizable para discernir qué reducción concreta ha de aplicarse.

Está claro que de alguna manera este autómata, en una versión simplificada, estaba ya presente como alternativa a la solución del problema de decidir qué regla reducir en el caso de precedencia débil en su forma original. Además, nuestro autómata ha de incorporarlo, pues en los métodos de precedencia siempre habremos de reconocer el handle completo para asegurar que las reducciones que vamos realizando son sintácticamente correctas. No vamos a entrar en el detalle interno de la construcción del autómata por las razones que ahora se verán.

Es importante considerar que, en la mayoría de los casos es posible conocer qué regla reducir tras leer muy pocos símbolos, y sólo en casos muy particulares y muy poco corrientes en gramáticas para lenguajes formales utilizables por personas, como las de lenguajes de programación—que por otro lado permitirían notables simplificaciones adicionales—será necesario hacer una exploración más profunda en la pila. En definitiva, no sólo el autómata se prevé que sea considerablemente pequeño, especialmente en comparación con su equivalente directo LR(1), sino que su utilización más frecuente incurrirá en leer (casi) sólo hasta el comienzo del handle que vamos a reducir.

En cualquier caso, podemos hacernos una idea de la disminución de tamaño en relación al autómata directo si consideramos que éste ha de *reconocer* el lenguaje completo de posibles contenidos de la pila más prelectura, mientras que nuestro autómata precisa sólo reconocer las colas—normalmente de muy pequeña longitud, como hemos dicho—de dicho lenguaje que nos permiten discernir entre distintas reducciones.

De esta forma, podríamos decir que disponemos de la potencia LR(1) *para reducir*. Hemos llegado a un resultado importante—potencia global próxima a LR(1)—mediante la combinación de las técnicas ya expuestas. Sin embargo, hay una cierta inelegancia, un cierto desequilibrio, en el resultado global que nos hizo reflexionar. Fundamentalmente se trata, por un lado, de la transformación, aunque fácilmente automatizable, pero siempre fastidiosa de reglas para solventar conflictos de precedencia, y, por otro, de la descompensación entre la sencillez del mecanismo de relaciones de precedencia para decidir si avanzar o reducir y la relativa complejidad conceptual del autómata para decidir qué reducción aplicar.

La idea que surgió fue: ¿por qué no utilizar el autómata para el problema completo, en lugar de sólo para una parte? ¿No será quizá más simple que la adición de los dos métodos? De alguna manera, el autómata podría englobar el cálculo de precedencia, si el análisis que realiza partiera, no de la cima de la pila, sino del símbolo de prelectura[1]. Aún más, puesto que haríamos uso de la información completa de la pila más prelectura, tendríamos en teoría completa potencia LR(1), eliminaríamos las inelegancias anteriores, y mantendríamos prácticamente en su totalidad la ventaja del pequeño tamaño.

4 Un autómata discriminante

La idea básica parte de que una gramática LR(1) permitirá siempre que un reconocedor ascendente pueda decidir, conociendo únicamente el contenido de la pila y el símbolo de la ventana actual, qué acción tomar a continuación: avanzar, reducir de acuerdo con alguna regla, o indicar un error. Por otro lado, es sabido que el contenido de la pila será siempre una sentencia de un lenguaje regular, por lo que será posible construir, para toda gramática LR(1) un autómata reconocedor.

[1]Como de hecho así ocurría; aunque aquí, por mayor sencillez en la exposición lo hemos presentado como comenzando desde la cima.

El método aquí propuesto, por tanto, consiste en *iniciar* la construcción de un autómata que reconozca el lenguaje de los contenidos de la pila, pero calculando para cada estado las diferentes acciones compatibles con el mismo, de forma que, si la acción fuera única, el algoritmo de construcción se para en esta parte del autómata. Es justamente este aspecto el que nos va a permitir obtener drásticas reducciones de tamaño en comparación con los métodos clásicos. Para obtener justamente la potencia LR(1) se ha de considerar, obviamente, el símbolo de la ventana: efectivamente, en cada estado se analiza la compatibilidad combinada de la cola de pila ya leída con el símbolo de la ventana, hasta que se halle una única acción por estado.

4.1 Utilización para el análisis

En cada paso de análisis, la exploración de la pila comienza desde el estado inicial partiendo desde la cima. En cada estado se utiliza en primer lugar el símbolo de la ventana para tomar una decisión; si ello no fuera posible, se utiliza el siguiente—en el sentido desde la cima hacia la base—símbolo de la pila para tomar una decisión o, en su defecto, una transición a un nuevo estado. De hecho, hemos podido comprobar que, aunque en teoría pudiera parecer que el método da lugar a repetidas y profundas exploraciones de la pila, en la práctica no es así.

La exploración finaliza con dos posibilidades: bien se decide una acción (el análisis termina si ésta es reducir al símbolo principal aumentado), bien ésta no puede decidirse debido a un error presente en la entrada. Esta última condición se puede detectar porque el símbolo siguiente en la pila no presenta transición legal en el estado actual, o porque se llega a la base de la pila sin haber alcanzado una decisión.

5 Resultados experimentales

De acuerdo con el método presentado, se ha construido un generador de analizadores sintácticos. Los resultados prácticos coinciden con la potencia de análisis y con los tamaños de autómata previstos. Como aún no disponemos de un generador clásico LR(1) con el que hacer comparaciones a igualdad de potencia, hemos elegido para realizar las comparaciones el bien conocido generador Yacc [10]. A pesar de que éste genera autómatas mucho menores que los correspondientes LR(1), las comparaciones experimentales han sido incluso favorables en todos los casos, y del orden de un 30 a un 40 por ciento menores en número de estados de los correspondientes autómatas. Podemos citar que, en particular, la gramática mayor que ha sido posible comparar, una gramática para el lenguaje C completo obtenida de la base de datos del grupo de noticias `comp.compilers`, adaptada para que fuera exactamente LALR(1), dio el siguiente resultado: con 235 reglas, Yacc generó 439 estados, mientras que nuestro autómata únicamente 284 estados.

6 Conclusiones

Se han presentado brevemente las bases teóricas que nos han conducido al desarrollo de un método para la construcción y utilización de un autómata que, recorriendo la pila desde la cima hacia la base, es capaz de decidir la siguiente acción sintáctica a tomar, en combinación con el símbolo de la ventana de prelectura.

Ello ha permitido obtener un nuevo analizador sintáctico para la clase de gramáticas LR(1). Dado que el método de construcción es discriminante, terminándose el proceso de construcción de estados desde el primer momento en que ello es posible, se han conseguido obtener muy importantes reducciones de tamaño en comparación con los métodos clásicos LR(1), manteniéndose los aspectos negativos dentro de márgenes razonables, poco relevantes desde un punto de vista práctico.

Desde un punto de vista teórico, se generaliza la idea de precedencia, aprovechando las ventajas que la intuición indicó a los primeros diseñadores de éstos métodos, abriéndose una nueva vía inexplorada de interesantes posibilidades teóricas y prácticas.

Agradecimientos

Este trabajo debe mucho a discusiones con Olivier Lecarme y Jacque Farré, del Laboratorio I3S del CNRS y la U. Niza-Sophia Antipolis (Francia), y al acogedor entorno del laboratorio. Desarrollado gracias a una ayuda del Gobierno y la Caja de Canarias. Quisiera reconocer especialmente a mi esposa, María del Mar.

References

[1] A. V. Aho and J. D. Ullman. *The Theory of Parsing, Translation and Compiling.* Prentice-Hall, 1973.

[2] G. V. Cormack. An LR substring parser for noncorrecting syntax error recovery. *ACM SIGPLAN Notices*, 24(7):161–169, July 1989.

[3] F. DeRemer. *Practical Translators for LR(k) Languages.* PhD thesis, M.I.T., Cambridge, Massachusetts, U.S.A., 1969.

[4] R. W. Floyd. Syntactic analysis and operator precedence. *J. ACM*, 10(3):316–333, 1963.

[5] R. W. Floyd. Bounded context syntactic analysis. *Comm. ACM*, 7(2):62–67, 1964.

[6] J. Fortes Gálvez. Generating LR(1) parsers of small size. In *International Workshop on Compiler Construction*, Paderborn, Germany, 1992. To be published in *Lecture Notes in Computer Science.*

[7] R. M. Graham. Bounded context translation. In *AFIPS Spring Joint Computer Conference*, pages 184–205, 1964.

[8] S. L. Graham and S. P. Rhodes. Practical syntactic error recovery. *Comm. ACM*, 18(11):639–650, 1975.

[9] J. D. Ichbiah and S. P. Morse. A technique for generating almost optimal Floyd-Evans productions for precedence grammars. *Comm. ACM*, 13(8):501–508, 1970.

[10] S. C. Johnson. Yacc—yet another compiler compiler. Computing Science Technical Report 32, AT&T Bell Laboratories, Murray Hill, N.J., 1975.

[11] D. E. Knuth. On the translation of languages from left to right. *Information and Control*, 8(6):607–639, 1965.

[12] W. M. McKeeman, J. J. Horning, and D. B. Wortman. *A Compiler Generator.* Prentice-Hall, 1970.

[13] M. T. Milani. On the descriptive power of simple precedence grammars. *Intern. J. Computer Math.*, 39:29–49, 1991.

[14] M. T. Milani and D. A. Workman. Epsilon weak precedence grammars and languages. *Informatique théorique et Applications/Theoretical Informatics and Applications*, 24(3):241–266, 1990.

[15] C. Pair. Trees, pushdown stores and compilation. *RFTI—Chiffres*, 7(3):199–216, 1964.

[16] H. Richter. Noncorrecting syntax error recovery. *ACM Transactions on Programming Languages and Systems*, 7(3):478–489, July 1985.

[17] S. Sippu and E. Soisalon-Soininen. *Parsing Theory.* Springer-Verlag, 1990.

[18] N. Wirth and H. Weber. EULER: A generalization of ALGOL and its formal definition. Parts I and II. *Comm. ACM*, 9(1):13–23 and 89–99, 1966.

PERSEO:
AMBIENTE DE PROGRAMACION DE APLICACIONES TRANSACCIONALES DISTRIBUIDAS SOBRE BASES DE DATOS RELACIONALES

Consuelo FRANKY
José ABASOLO
Raúl CUCALON
Gloria ACOSTA
Santiago MAYA

Depto. de Ingeniería de Sistemas y Computación
Universidad de los Andes
Apartado Aéreo 4976
e-mail: cfranky@andescol.bitnet
Santafé de Bogotá- COLOMBIA

RESUMEN:

En este artículo se presenta la descripción de un proyecto en desarrollo conducente a un conjunto de facilidades para la programación de aplicaciones transaccionales distribuidas sobre bases de datos relacionales. Dichas bases de datos se suponen ubicadas en diversos computadores conectados por medio de una red de comunicación. Sobre la plataforma que constituye el conjunto disponible de sistemas manejadores de las bases de datos de una organización cualquiera, se construye un conjunto de servicios con una interfaz para el programador de aplicaciones distribuidas. El ambiente de programación resultante, "PERSEO", ofrece transparencia total a la fragmentación, a la distribución de las bases de datos y a la heterogeneidad de los sistemas manejadores relacionales presentando al programador la visión de un solo sistema centralizado. Para ello el ambiente asegura automáticamente las funciones relacionadas con la descomposición y optimización de consultas distribuidas. Además libera al programador del manejo complejo de transacciones distribuidas, incluyendo recuperación ante fallas y control de concurrencia, funciones todas que asegura el ambiente. Algunos de los aspectos ofrecidos por PERSEO no se encuentran aún disponibles en los sistemas manejadores de bases de datos distribuidas a nivel comercial, hecho que lo convierte en un proyecto novedoso y útil.

PALABRAS CLAVES: Bases de Datos Distribuidas. Sistemas transaccionales distribuidos. Programación de aplicaciones distribuidas sobre Bases de Datos.

1. INTRODUCCION

Indudablemente las redes de comunicación son en la actualidad un avance que se afianza cada vez más en los diferentes ámbitos de las organizaciones. El establecimiento de redes nacionales e internacionales impulsa el desarrollo de aplicaciones distribuidas, especialmente en aquellas organizaciones de carácter netamente distribuido que poseen departamentos o seccionales geográficamente dispersas y que requieren de la combinación de sus diferentes bases de datos para operar adecuadamente.

Sin embargo, la programación de aplicaciones distribuidas sobre el conjunto de bases de datos de una organización puede resultar una labor muy compleja si no se cuenta con ayudas adecuadas. Una primera alternativa para facilitar la labor del programador es contar con un ambiente basado en un Sistema Manejador de Bases de Datos Distribuidas que opere sobre el conjunto de computadores que componen la red de la organización. La adquisición de un ambiente de esta naturaleza, por parte de una organización, puede implicar tener que adquirir también sistemas manejadores locales y la conversión de bases de datos existentes para que puedan ser manejadas en el nuevo ambiente, resultando todo ello en un proceso muy costoso.

Una segunda alternativa menos onerosa es aprovechar la plataforma existente en la organización, referente al conjunto de sistemas manejadores de bases de datos locales, y desarrollar sobre esta plataforma un conjunto de servicios con una interfaz para el programador. Esta segunda alternativa es la que nosotros ofrecemos en el ambiente **PERSEO**: en este proyecto (que actualmente se encuentra en desarrollo) pretendemos elaborar librerías de funciones que el programador pueda incluir e invocar en sus aplicaciones con el fin de brindarle transparencia a la fragmentación y distribución de las bases de datos, con el resultado de que la programación resulta similar a la que se tiene en un solo sistema centralizado de bases de datos. Se trata entonces de aprovechar las facilidades que brindan los sistemas manejadores de bases de datos existentes en los diferentes computadores de la organización y por otro lado evitar la conversión de las bases de datos existentes.

El conjunto de funciones que constituyen el ambiente PERSEO facilita la programación de aplicaciones transaccionales distribuidas sobre un conjunto de bases de datos relacionales localizadas en computadores conectados a través de una red de comunicación. Las principales facilidades que ofrece PERSEO son las siguientes:

- **Transparencia a la fragmentación y a la distribución de las bases de datos relacionales**: la visión que tiene el programador es la de un conjunto de tablas globales relacionales ubicadas en un solo sistema centralizado.

- **Transparencia a la heterogeneidad de los sistemas manejadores relacionales**: el programador invoca dentro de su aplicación funciones tipo C que le proveen servicios de consulta y actualización sobre el conjunto de tablas globales que constituyen el sistema. Estas funciones son equivalentes en su expresividad al lenguaje SQL y constituyen una interfaz estándar a un conjunto de bases de datos relacionales administradas por diversos sistemas manejadores relacionales.

- **Descomposición y optimización de consultas y actualizaciones**: PERSEO provee funciones que transforman pedidos de consultas y actualizaciones expresadas por el programador en términos de tablas globales a conjuntos de operaciones sobre tablas locales en bases de datos específicas. En el caso de consultas se busca además minimizar el volumen de datos a transferir (mediante la selección óptima de los sitios encargados de realizar operaciones binarias como JOIN's entre tablas ubicadas en distintas bases de datos).

- **Ejecución de consultas y actualizaciones**: PERSEO se encarga de la transmisión de mensajes necesarios para la ejecución de una consulta o actualización que involucra múltiples bases de datos. Además sincroniza los procesos participantes y reúne los resultados que se le van a retornar a la aplicación. Para asegurar transparencia a la heterogeneidad de los diferentes sistemas manejadores relacionales, los mensajes llevan pedidos en términos de una interfaz estándar que luego debe transformarse en cada sitio al lenguaje SQL específico de la base de datos local.

Manejo de transacciones distribuidas: el programador puede invocar en su aplicación funciones especiales para delimitar el principio y el final de una transacción, la cual incluye en general varios pedidos de consultas y actualizaciones sobre las múltiples bases de datos que constituyen el sistema. PERSEO maneja automáticamente la ejecución simultánea de múltiples transacciones distribuidas solicitadas en diferentes sitios, liberando al programador de tener que prever funciones de recuperación ante fallas para asegurar la atomicidad de cada transacción y de control de concurrencia en el acceso a datos comunes (incluyendo el tratamiento de abrazos mortales distribuidos) : todas estas funciones las provee el ambiente.

La suma de facilidades que ofrecen PERSEO y los sistemas manejadores relacionales en los que se apoya son equivalentes a las facilidades que ofrecería un sistema manejador de bases de datos distribuidas. La novedad de este proyecto radica en el hecho de que los sistemas manejadores de bases de datos distribuidas disponibles actualmente a nivel comercial, no ofrecen todavía los aspectos relativos a fragmentación y al manejo completo de transacciones distribuidas. Desde el punto de vista del grado de evolución de los sistemas manejadores de bases de datos distribuidas presentado en [10] PERSEO corresponde al cuarto y máximo grado en razón de que maneja transacciones distribuidas compuestas de pedidos, en donde cada uno de ellos puede involucrar múltiples bases de datos de diferentes sitios.

El enfoque adoptado en PERSEO consistente en presentar al programador una interfaz de acceso a una Bases de Datos Distribuida a través de un conjunto de funciones C es también novedoso y atractivo. En efecto, este enfoque es más uniforme y más simple de usar que el enfoque tradicional de incluir claúsulas SQL mezcladas con instrucciones C en un programa de aplicación. Además permite independizar la aplicación de los diferentes sistemas manejadores relacionales participantes.

A continuación en la sección 2 de este artículo se presentan las facilidades de especificación de esquemas que ofrece PERSEO al administrador de un sistema de múltiples bases de datos. En la sección 3 se presenta la interfaz de servicios ofrecidos en PERSEO al programador de aplicaciones distribuidas sobre múltiples bases de datos. En la sección 4 se muestra la arquitectura del sistema PERSEO en términos de procesos Clientes - Servidores, con sus módulos asociados, encargados de ejecutar transacciones distribuidas solicitadas por las aplicaciones. En la sección 5 se describe el módulo Optimizador que provee las funciones de descomposición y optimización de consultas y actualizaciones distribuidas. En la sección 6 se describen los módulos encargados de coordinar y ejecutar transacciones distribuidas. Por último, en la sección 7 se plantean las proyecciones y futuras etapas del proyecto PERSEO.

2. FACILIDADES DE ESPECIFICACION DE ESQUEMAS QUE SE OFRECEN AL ADMINISTRADOR DE UN SISTEMA DE MULTIPLES BASES DE DATOS

El ambiente PERSEO provee un programa Utilitario que permite a un administrador de un sistema de múltiples bases de datos relacionales especificar los esquemas global, de fragmentación y de distribución. Siguiendo las definiciones de [3] el administrador especificará los esquemas bajo el modelo relacional con las siguientes opciones:

- Esquema Global: corresponde a especificar el esquema relacional de tablas globales que componen el conjunto de bases de datos. Este esquema corresponde a la visión centralizada del sistema que se le quiere dar al programador de aplicaciones (visión denominada base de datos global).

- Esquema de Fragmentación: el administrador podrá especificar una fragmentación horizontal (simple y/o derivada hasta un nivel de profundidad) de las tablas globales. En la primera versión de PERSEO no se soportará la fragmentación vertical ni la combinación de la fragmentación horizontal con la vertical.

- Esquema de Distribución: corresponde a especificar el sitio de localización de cada uno de los fragmentos de las tablas globales. En la primera versión de PERSEO no se soportará la replicación de fragmentos.

El programa Utilitario materializa las anteriores especificaciones en un Catálogo Global del sistema que debe replicarse en cada uno de los sitios para poder llevar a cabo las funciones de descomposición y optimización de consultas y de actualizaciones distribuidas.

Adicionalmente, el programa Utilitario permite actualizar el Catálogo Global cuando hay creación y destrucción de tablas e igualmente permite mantener y actualizar estadísticas sobre las tablas con miras a proveer información más precisa a las funciones de optimización.

3. INTERFAZ DE SERVICIOS OFRECIDA AL PROGRAMADOR DE APLICACIONES DISTRIBUIDAS SOBRE BASES DE DATOS

El programador de aplicaciones distribuidas sobre un conjunto de bases de datos relacionales cuenta en PERSEO con una librería de funciones tipo C que le ofrece transparencia a la fragmentación, distribución y heterogeneidad de los sistemas manejadores relacionales. Adicionalmente cuenta con funciones que le permiten delimitar transacciones distribuidas (que no deben estar anidadas). En total son 4 las funciones de PERSEO que el programador puede invocar dentro de su aplicación (la cual debe escribir en lenguaje ANSI C [11]):

- **begin_transaction**: con la invocación de esta función la aplicación señala el inicio de una transacción distribuida que contendrá solicitudes de consultas y actualizaciones sobre tablas globales intercaladas con operaciones locales (i.e. interacción con el usuario y manipulación de los datos resultantes de las consultas).

- **end_transaction**: con la invocación de esta función la aplicación señala el fin de una transacción distribuida. El sistema asegurará la atomicidad de la transacción completando sus efectos sobre los datos involucrados, o anulando esos efectos en caso de que haya problemas como caídas de sitios participantes o conflictos de concurrencia.

- **sql**: con la invocación de esta función la aplicación solicita una consulta o actualización sobre tablas globales. El único parámetro de esta función es una cadena con la sentencia SQL solicitada, la cual puede incluir las operaciones SELECT, INSERT o DELETE. La operación SELECT podrá usar funciones agregadas (i.e. AVG, SUM, COUNT, etc.) así como cláusulas de ordenamiento (GROUP BY, HAVING).

  En general la función sql permite a la aplicación solicitar cualquier operación del lenguaje estándar SQL [10] excepto la operación UDPDATE que será tratada en una siguiente versión del proyecto.

- **sqlv**: esta función permite también solicitar una consulta o actualización sobre tablas globales y adicionalmente permite usar en la sentencia SQL parámetros variables cuyos valores se conocen solo en tiempo de ejecución. Un parámetro de esta función sqlv podrá indicar opcionalmente una función que la aplicación quiere ejecutar sobre cada una de las tuplas resultantes de una consulta (por ejemplo una función para imprimir dichas tuplas o hacer cálculos con ellas) : esta función debe estar definida dentro de la aplicación.

En resumen, la aplicación escrita por el programador se ocupa de la interacción local con el usuario mientras que el manejo de todos los aspectos distribuidos del sistema corre por cuenta de las funciones provistas por el ambiente PERSEO. Es de esperar que todos los accesos a las bases de datos del sistema, incluyendo la local, se hagan a través de la interfaz de servicios descritos en esta sección. Esta interfaz usada por la aplicación se denomina "IABDD" (Interfaz de Acceso a Bases de Datos Distribuidas) y se diferenciará de la interfaz "IABD" (Interfaz de Acceso a una Base de Datos) usada por los procesos del sistema para accesar una base de datos particular con transparencia a la heterogeneidad de los diferentes sistemas manejadores relacionales que participan en el sistema.

4. ARQUITECTURA DEL SISTEMA PERSEO

Una aplicación distribuida sobre múltiples bases de datos que invoque los servicios de PERSEO a través de la interfaz IABDD, da lugar en ejecución a un proceso CLIENTE que va a interactuar con procesos SERVIDORES provistos por el sistema.

Conceptualmente un proceso CLIENTE está compuesto por 3 módulos:

- la **aplicación** propiamente dicha: Interactúa con el usuario e invoca los servicios distribuidos de PERSEO a través de la interfaz IABDD (Interfaz de Acceso a Bases de Datos Distribuidas).

- el módulo **Coordinador**: Sincroniza la ejecución de los servicios distribuidos invocados por la aplicación. Para ello invoca un módulo Optimizador que se encarga de elaborar el plan de ejecución de cada consulta o actualización distribuida, y luego se sincroniza con los procesos SERVIDORES del sistema para llevar a cabo dicho plan. Igualmente se sincroniza con los procesos SERVIDORES para asegurar la ejecución consistente de la transacción distribuida solicitada por la aplicación.

- el módulo **Optimizador**: Descompone cada solicitud de consulta o actualización de tablas globales en conjuntos de instrucciones que el Coordinador debe dirigir a bases de datos específicas. La descomposición corresponde a una transformación de una solicitud de servicio expresada en interfaz IABDD (Interfaz de Acceso a Bases de Datos Distribuidas) a un conjunto de solicitudes de servicios expresadas en interfaz IABD (Interfaz de Acceso a una Base de Datos) destinadas a sitios específicos. Para llevar a cabo esta función, el Optimizador se basa en la información sobre esquemas de la base de datos global contenida en el Catálogo Global y utiliza como criterio de optimización minimizar el volumen de datos a transferir (criterio adecuado para sistemas geográficamente distribuidos [3]).

En términos de implantación, el proceso CLIENTE proviene de la compilación del programa de aplicación junto con librerías provistas por el sistema, las cuales corresponden a los módulos Coordinador y Optimizador.

Cada proceso SERVIDOR interactúa con el sistema manejador relacional de una base de datos local para ejecutar las instrucciones provenientes de los procesos CLIENTES del sistema (cada CLIENTE está asociado a una transacción distribuida en curso de ejecución). Los pedidos de las instrucciones llegan expresados en interfaz IABD que es independiente de cualquier base de datos local. El SERVIDOR debe entonces transformar cada pedido en intrucciones SQL propias del sistema manejador relacional local y para ello utiliza una librería IABD particular para cada sistema manejador. Adicionalmente ejecuta instrucciones de control asociadas a la ejecución consistente de transacciones distribuidas.

La figura 1 muestra la arquitectura del sistema PERSEO en términos de procesos CLIENTES - SERVIDORES y de sus módulos asociados. También se ilustra la ubicación de las interfaces IABDD (aplicación - PERSEO) e IABD (SERVIDOR - sistema manejador local).

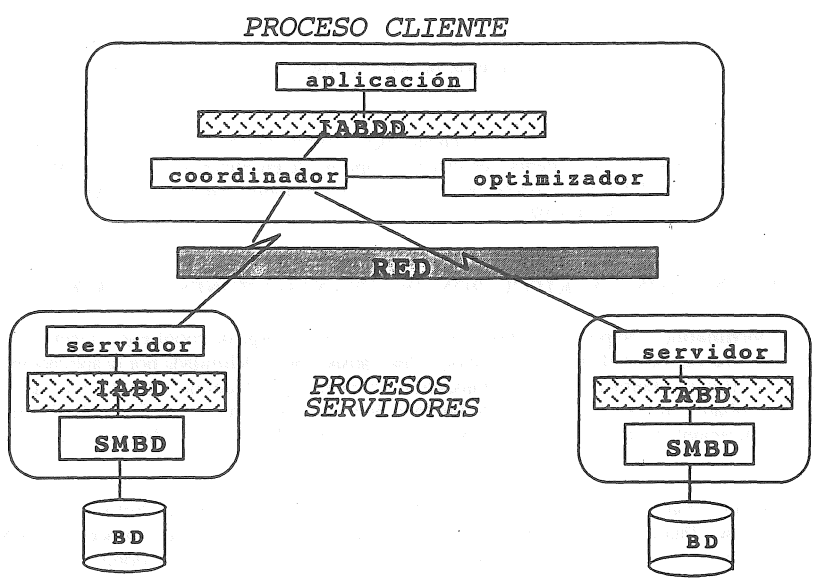


Figura 1. Arquitectura del sistema PERSEO.

Para múltiples transacciones simultáneas se tendría un proceso CLIENTE en el sitio origen de cada transacción. Por otro lado hay un proceso SERVIDOR en cada sitio que maneje una base de datos del sistema. Un proceso CLIENTE requiere comunicarse en general con procesos SERVIDORES remotos y también con el proceso SERVIDOR local.

5. DESCOMPOSICION Y OPTIMIZACION DE CONSULTAS Y ACTUALIZACIONES DISTRIBUIDAS

El módulo Optimizador, componente del proceso CLIENTE, se encarga de descomponer y optimizar las consultas y actualizaciones distribuidas que se quieran realizar sobre las múltiples bases de datos que componen el sistema.

En el ambiente PERSEO, cada pedido de consulta o actualización sobre tablas globales, hecho por una aplicación, genera una invocación al módulo Optimizador, a quien se le informa como parámetro una cadena con la sentencia SQL que se quiere ejecutar. En la primera versión de PERSEO la interfaz IABDD permite al programador hacer pedidos SQL que incluyen operaciones SELECT, INSERT y DELETE, como se explicó en la sección 3 (Interfaz IABDD de servicios ofrecidos al programador).

Con base en la información de esquemas (global, de fragmentación y de distribución) contenida en el catálogo global, el Optimizador genera el plan óptimo de ejecución de la sentencia SQL en el cual identifica los SERVIDORES que deben participar y las operaciones que cada uno de ellos debe ejecutar. Estas operaciones se expresan en términos de la interfaz IABD, independiente de cualquier sistema manejador relacional. El criterio de optimización, como ya se mencionó, es minimizar el volumen de datos a transferir por la red.

Para generar el plan óptimo de ejecución de una sentencia SQL, el módulo Optimizador cubre las siguientes etapas (basándose en la teoría de [3] y en algoritmos de tranformación SQL - algebra relacional [4])):

- transformación de la sentencia SQL en el árbol equivalente de operaciones del álgebra relacional sobre tablas globales (árbol global)

- transformación del árbol global en un árbol canónico expresado en términos de fragmentos

- optimización y materialización del árbol canónico mediante la eliminación de subpartes que conducen a resultados nulos y mediante la selección de sitios óptimos para realizar las operaciones binarias JOIN's

- transformación del árbol óptimo en sentencias SQL expresadas en interfaz IABD y destinadas a SERVIDORES específicos. Además, en este plan óptimo de ejecución se incluyen operaciones que deben realizar los SERVIDORES referentes a la creación de tablas temporales, envío y recepción de las mismas por otros SERVIDORES.

6. COORDINACION Y EJECUCION DE TRANSACCIONES DISTRIBUIDAS

El módulo **Coordinador**, componente del proceso CLIENTE, se encarga de coordinar la ejecución de cada transacción distribuida solicitada por la aplicación. Una transacción es delimitada por la aplicación a través de las funciones Begin_Transaction y End_Transaction y contiene solicitudes de consultas y actualizaciones sobre tablas globales (invocadas a través de las funciones sql y sqlv de la interfaz IABDD) junto con instrucciones puramente locales (interacción con el usuario, manipulación de los datos que resultan de una consulta, etc).

La ejecución de una transacción distribuida en el ambiente PERSEO se rige por el Protocolo de Validación en dos etapas ("Two Phase Commit Protocol") [6] y exige la sincronización entre el proceso CLIENTE del sitio donde se origina la transacción y los procesos SERVIDORES de los sitios en donde residen las bases de datos involucradas por la transacción.

Las siguientes son las principales funciones que realiza el módulo Coordinador en relación a cada transacción distribuida solicitada por la aplicación:

- **Generación de una estampilla** para identificar la transacción de forma única respecto a las demás transacciones distribuidas del sistema.

- **Distribución del plan óptimo de ejecución de una consulta o actualización**: Después de invocar al módulo Optimizador se obtiene el plan de ejecución de una consulta o actualización sobre tablas globales. El módulo Coordinador envía a cada uno de los SERVIDORES participantes el conjunto de operaciones del plan que debe ejecutar, junto con la estampilla de la transacción.

- **Aplicación de funciones a los resultados de las consultas globales**: en caso de que el programa de aplicación especifique una función en una solicitud de consulta global, el módulo Coordinador la aplicará a cada una de las tuplas que son resultado de la consulta antes de devolver el control a la aplicación propiamente dicha.

- **Atomicidad de la transacción**: Ante una solicitud de End_Transaction hecha por la aplicación, el módulo Coordinador realiza las etapas de validación de la transacción enviando los mensajes correspondientes a los SERVIDORES participantes. Bajo el Protocolo de Validación en dos etapas, el proceso CLIENTE origen de la transacción debe lograr que todos los SERVIDORES participantes completen los efectos de la transacción ("commit") o todos los anulen ("rollback") a fin de asegurar la atomicidad de la transacción (aún en caso de fallas).

- **Control de concurrencia y prevención de abrazos mortales entre transacciones** : Para asegurar el control de concurrencia entre transacciones que accesan datos comunes, cada transacción utiliza candados para reservar las tablas que requiere (cumpliendo la propiedad de reservación en "dos fases" [3]). Para resolver situaciones de abrazos mortales entre dos o más transacciones, en PERSEO se aplica una técnica de prevención de abrazos mortales.

En cada uno de los sitios del sistema se tiene un proceso SERVIDOR capacitado para ejecutar sobre la base de datos local diversas solicitudes provenientes de múltiples transacciones distribuidas.

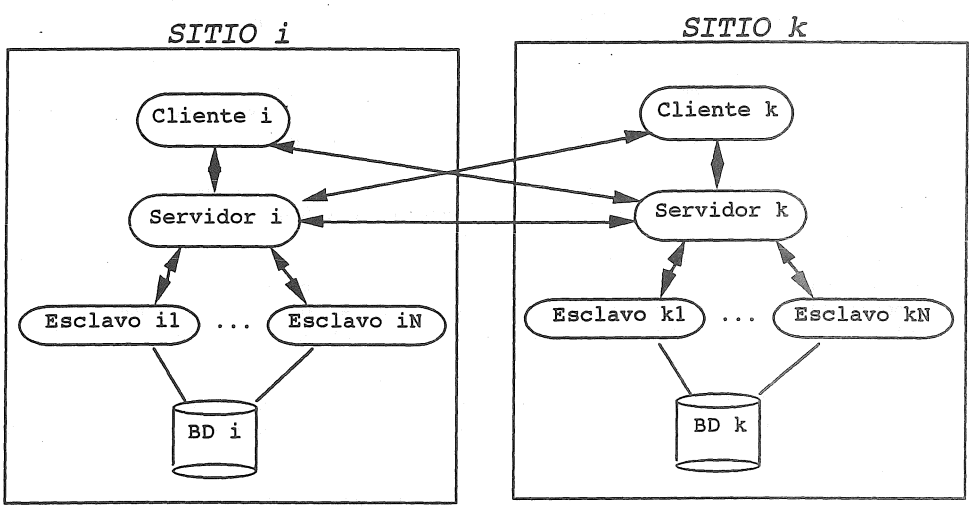


Figura 2. Relación de comunicación entre procesos CLIENTES, SERVIDORES y ESCLAVOS en el sistema PERSEO.

Para asegurar una buena eficiencia en la atención de las solicitudes de las transacciones, el proceso SERVIDOR activa un proceso ESCLAVO por cada transacción en curso que involucre su sitio. De esta

manera se pueden atender simultáneamente múltiples solicitudes de transacciones distintas. El proceso SERVIDOR centraliza, sin embargo, la función de control de concurrencia entre esas transacciones basándose en el manejo de candados. Al finalizar una transacción, finalizan también sus procesos ESCLAVOS activados en los sitios participantes. Son los procesos ESCLAVOS y no los SERVIDORES quienes utilizan una librería IABD (Interfaz de Acceso a una Base de Datos) específica para el sistema manejador relacional local.

La figura 2 ilustra la relación de comunicación entre procesos CLIENTES, SERVIDORES y ESCLAVOS correspondientes a diversas transacciones.

El plan global de una consulta o actualización invocada por una aplicación da lugar a solicitudes de operaciones sobre bases de datos específicas y a solicitudes de envíos o recepciones de tablas temporales que resultan de ejecutar dichas operaciones. Por esta razón, el proceso SERVIDOR debe estar en capacidad de enviar la tabla temporal resultante de una operación de consulta sobre la base de datos local hacia otro sitio que requiera dicha tabla. Igualmente el proceso SERVIDOR debe estar en capacidad de recibir una tabla temporal y avisarle al proceso ESCLAVO que la esté esperando.

En el Ambiente PERSEO se aprovechan las facilidades de los sistemas manejadores relacionales disponibles en los sitios con el fin de asegurar la ejecución consistente de transacciones distribuidas. Por ejemplo, para completar los efectos de una transacción en un sitio, el proceso ESCLAVO asociado invocará la operación "Commit" o "Rollback" que le ofrece el sistema manejador local. Sin embargo, ciertas funciones como el control de concurrencia de transacciones distribuidas, requieren replicar funciones de los sistemas manejadores: en este caso, es necesario que cada proceso SERVIDOR registre los candados que diversas transacciones ponen sobre las tablas de la base de datos local a fin de poder prevenir abrazos mortales distribuidos (el manejo de candados por parte de un sistema manejador permite prevenir o detectar abrazos mortales únicamente locales pero no distribuidos).

7. PROYECCIONES Y EXTENSIONES PREVISTAS

El proyecto PERSEO se desarrolla actualmente sobre una red local de computadores UNIX comunicados bajo protocolos TCP/IP [5]. La primera versión debe soportar aplicaciones distribuidas sobre bases de datos relacionales ORACLE y UNIFY ubicadas respectivamente en un computador Unysis U6000 y en un computador Digital MicroVax II. Para lograr el acceso transparente a la heterogeneidad de los sistemas manejadores relacionales se utilizan librerías que soportan la interfaz IABD para ORACLE y para UNIFY desarrolladas en proyectos paralelos ([2], [12]).

La comunicación remota entre procesos CLIENTES y SERVIDORES utiliza la interfaz de sockets sobre protocolos TCP/IP y la comunicación local CLIENTE - SERVIDOR y SERVIDOR - ESCLAVOS utiliza el mecanismo de Colas de Mensajes ("Messages Queues") propio de UNIX [16].

Para independizar el proyecto PERSEO de los mecanismos específicos que se utilicen para la comunicación entre procesos, su segunda versión se implantará en el lenguaje y ambiente JOYCE+ [8] : este lenguaje es una extensión de C que permite una programación de alto nivel de los procesos de un sistema distribuido, de manera independiente a la red de comunicación en donde van a operar. El ambiente provisto para el lenguaje JOYCE+ [1] ofrecerá todos los mecanismos necesarios para brindar transparencia al programador respecto a la red específica en donde se va a trabajar. Actualmente el lenguaje JOYCE+ se utiliza en el proyecto PERSEO como lenguaje de especificación de los procesos CLIENTES, SERVIDORES y ESCLAVOS que lo componen.

Las ideas principales del proyecto PERSEO provienen de un proyecto anterior "TESEO" ([7], [13], [14]) que fue implantado sobre una red de microcomputadores que presentaba facilidades muy restringidas de manejo de bases de datos.

Como versiones futuras del proyecto PERSEO se planea el tratamiento de la operación SQL "UPDATE", el manejo de tablas replicadas, el manejo dinámico del Catálogo Global y herramientas que faciliten el diseño de los esquemas de fragmentación y de distribución de la base de datos global. Igualmente se buscará implantar herramientas que faciliten la integración de bases de datos heterógeneas. Estos objetivos plantean retos novedosos teniendo en cuenta que casi ninguna de estas facilidades se ofrecen actualmente en

los sistemas manejadores de bases de datos distribuidas disponibles a nivel comercial: de hecho, se observa un gran vacío en lo que respecta a ayudas al diseño.

En conclusión, con el proyecto PERSEO pretendemos crear un Ambiente que facilite la programación de aplicaciones distribuidas sobre bases de datos, aprovechando facilidades de sistemas manejadores relacionales existentes en distintos sitios y sin requerir la inversión costosa de adquirir o construir todo un sistema manejador de bases de datos distribuidas.

En las versiones futuras del proyecto PERSEO buscaremos crear un Ambiente que apoye cada una de las etapas del desarrollo de una aplicación distribuida sobre múltiples bases de datos, tanto lo que respecta al diseño de la distribución de los datos como al diseño de la distribución del procesamiento.

BIBLIOGRAFIA

[1]: A. Arenas S., C. Franky; "Ambiente JOYCE+ de desarrollo de sistemas distribuidos para entorno UNIX - TCP/IP", Memo CIFI de la Facultad de Ingeniería de la Universidad de los Andes, Bogotá-Colombia, 1991.

[2]: R. Blanco; "Interface de acceso a un sistema manejador de base de datos relacional desde un programa C", tesis de pregrado en Ingeniería de Sistemas y Computación, Universidad de los Andes, Bogotá - Colombia, 1992.

[3]: S. Ceri, G. Pelagatti; "Distributed Databases: Principles and Systems"; McGraw-Hill 1984.

[4]: S. Ceri, G. Gotlob; "Translating SQL into relational algebra: optimizations, semantics and equivalence of SQL queries", IEEE Transactions on Software Engineering, Vol. SE-11 N° 4, Abril de 1984.

[5]: D. E. Comer; "Internetworking with TCP/IP", Prentice-Hall 1988.

[6]: G. F. Coulouris, J. Dollimore; "Distributed Systems", Addison-Wesley 1988.

[7]: C. Franky ; "TESEO: Ambiente de programación de sistemas transaccionales distribuidos", XIII Conferencia Latinoamericana de Informática (CLEI), Bogotá - Colombia, Noviembre de 1987.

[8]: C. Franky.; "JOYCE+ : Model an Language for multi-site Distributed Systems", IEEE-ACM Second International Symposium on Databases in Parallel and Distributed Systems, Dublin Irlanda, Julio 1.990. También como Memo CIFI de la Facultad de Ingeniería de la Universidad de los Andes, 1990.

[9]: C. Franky; "Desarrollo de aplicaciones distribuidas sobre bases de datos", Revista ACIS de Sistemas, Bogotá - Colombia, Agosto de 1991.

[10] : J. R. Groff, P. N. Weinberg; "Aplique SQL", McGraw-Hill 1991.

[11]: B. Kernighan, D. Ritchie; "The C programming language", Prentice-Hall 1988.

[12] : C. Jiménez, R. Blanco; "IABD: Diseño e implantación de un API para el acceso a un sistema manejador de base de datos relacional desde un programa C", XVIII Conferencia Latinoamericana de Informática (CLEI), Las Palmas de Gran Canaria - España, Agosto de 1992.

[13]: P. Levy ; "Compilador de expresiones de lenguaje relacional para el sistema TESEO", XIII Conferencia Latinoamericana de Informática (CLEI), Bogotá - Colombia, Noviembre de 1987.

[14]: H. Sin ; "Un sistema de consulta para bases de datos distribuidas", tesis de pregrado en Ingeniería de Sistemas y Computación, Universidad de los Andes, Bogotá - Colombia, 1986.

[15]: M. Sloman, J. Kramer.; "Distributed Systems and Computer Networks", Prentice-Hall, 1987.

[16]: R. Stevens; "Unix Network programming", Prentice-Hall 1990.

Towards Efficient Partial Evaluation in Logic Programming

David A. Fuller, Sacha A. Bocic, Leo E. Bertossi

Computer Science Department
Pontificia Universidad Católica de Chile
PO Box 306
Santiago 22
Chile

Tel. +(56-2) 552 2375 x. 4440
Fax +(56-2) 552 4054

E-mail (internet): {dfuller, sacha, bertossi}@lascar.puc.cl

ABSTRACT: Partial evaluation is a symbolic manipulation technique used to produce efficient algorithms when part of the input to the algorithm is known. Other applications of partial evaluators such as universal compilation and compiler generation are also known to be possible. A partial evaluator receives as input a program and partially known input to that program, and outputs a residual program which runs at least as efficient as the input program with restricted input.

In this paper we study the case where both the input and residual programs are logic programs, being the partial evaluator itself a logic program. Up to now, partial evaluators have failed to produce large "non-toy" examples. Here we present extensions to the partial evaluation operations which will allow us to produce more efficient residual programs using less computing resources during partial evaluation. Examples are given.

KEYWORDS: Partial evaluation, logic programming, symbolic manipulation.

1. INTRODUCTION

Partial evaluation has been known as a program optimization technique since the early seventies [8, 3]. Only in the last few years, however, its use has been made possible. In this paper we will define extensions to partial evaluator which will made possible to optimize large programs when part of its input is known at compile time (i.e. partial evaluation time). We will also present some examples showing the power of this optimization technique. One of the examples will also describe a partial evaluator working as a universal compiler, producing efficient target programs.

Partial evaluation of a subject program wrt known values of some of its input parameters will result in a residual program. Running the residual program on any remaining input values will yield the same result as running the original subject program on all of the same input values. A residual program, therefore, is a specialization of a subject program with respect to known values of some of its parameters.

In expert systems and logic programming, this technique is now known as a way of eliminating the extra layers of interpretation produced by meta-interpreters [16, 15, 10].

In this section we will describe the equations defining partial evaluation and the meaning they have in the optimization of algorithms, automatic production of compilers and compiler generators. Let P be a programming language with a semantic function $\mathcal{P}$: `D → (D → D)`, where D is a set whose elements may represent programs in various languages, as well as their input and output. Note that $\mathcal{P}$ is a partial function, since programs may have infinite loops. A residual program r is defined by:

$$\mathcal{P}\texttt{(p) (<d1,d2>)} = \mathcal{P}\texttt{(r) (<d2>)}$$

where d1 is the known parameter, d2 the unknown, and p and r are valid programs in language P. Let *mix* and int be two valid programs in P, where *mix* is a partial evaluator, int is an interpreter for programs in language L, and l a valid program in L. The description of the three residual programs is given in the following equations:

target	=	$\mathcal{P}$(*mix*)(<int, l>)	(1.1)
comp	=	$\mathcal{P}$(*mix*)(<*mix*, int>)	(1.2)
cocom	=	$\mathcal{P}$(*mix*)(<*mix*, *mix*>)	(1.3)

The residual program target is generated by running the partial evaluator *mix* with input parameters int and l, and corresponds to the compiled program l into P. It should be noted that this is exactly how partial evaluation is being used in meta-interpretation. This equation also shows the way of using a partial evaluator as a program optimizer. In this case, program int represents the input program to be optimized, l corresponds to the partially known values to int, and target is the optimized program int wrt l.

The second equation shows the production of a compiler comp by running *mix* with the inputs *mix* and int. In other words, we are specializing the interpreter int for language L into a compiler for language L. A *mix* partial evaluator with the ability to apply to itself will be called a *mix* self-applicable partial evaluator. As explained in a latter section, not every partial evaluator has this ability.

The third equation describes how to produce the residual program cocom, a compiler-compiler or compiler generator, by running a *mix* self-applicable partial evaluator with its two inputs being copies of *mix* . Thus, when running cocom it will produce a compiler for a language L when an interpreter for language L is given.

We have briefly summarized the theory of partial evaluation and self-application. The first *mix* self-applicable partial evaluator was described in [9], using a first order subset of LISP. In the case of logic programming, a *mix* self-applicable partial evaluator for PROLOG programs was first described in [4, 5].

In this paper, we present an extended partial evaluator capable of undertaking powerful partial evaluation of logic programs. First, we present basic operations in a partial evaluator such as expansion and suspension of predicates during the process of building the proof tree. The following two sections introduce new operations to partial evaluators. Subsequently, a treatment of negated predicates during partial evaluation is presented. Then, local optimizations are introduced in order to optimize the residual program, and practical experience is presented. Finally, conclusions are drawn and further research is outlined.

2. BASIC OPERATIONS

In terms of logic programming, a partial evaluator *mix* attempts to build a proof tree of its first input program wrt its second one. For example, from Equation (1.1), *mix* will attempt to produce a proof tree for int running l. The proof tree thus produced will correspond to the residual target program. There are basic operations to build the proof tree, which will be discussed in this section.

2.1. Expansion

Expansion of a predicate p corresponds to building a partial proof tree for p [11]. The expansion of the user's goal will then produce the residual program.

This can be implemented easily in PROLOG with the help of a predicate `pe/2` which receives as first argument a predicate and returns in its second argument the proof tree for the predicate as a list. This is shown in Fig. 1.

Note that this implementation does not handle built-in predicates. In fact, if the variable `Goal` is instantiated to a built-in predicate, the first definition of `pe/2` will fail because `clause(Goal,true)` fails to find a solution for all predicates that are not facts in the PROLOG database. The second definition of `pe/2` also fails because `Goal` cannot unify with a term like `(Goal,Goals)`. The third definition fails because there is no definition for `Goal` in the database (since it is a built-in predicate that the user cannot redefine). However, extending the definition of `pe/2` to handle built-in predicates is trivial. This will be shown later. The implementation uses the PROLOG unification mechanism to unify variables of predicates that are being partially evaluated.

```
:- mode pe(+,-).
pe(Goal,[Goal]):- clause(Goal,true).[1]

pe((Goal,Goals),Tree):- pe(Goal,Tree1),
                        pe(Goals,Tree2),
                        append(Tree1,Tree2,Tree).

pe(Goal,Tree):- clause(Goal, Body),
                pe(Body,Tree1),
                append([(Goal:-Body)],Tree1,Tree).[2]
```

Figure 1. Expansion in partial evaluation.

1. `clause(H,B)` is a built-in predicate which is true if there is a clause `H:-B` in the PROLOG database. If `B` is the constant "true" then `H` is a fact.
2. `append/3` is true if its third argument is syntactically equal to the concatenation of the first two arguments.

An important fact is that the implementation only finds the first solution. Other solutions have to be found using backtracking or the built-in predicate `bagof`. At partial evaluation time we have less information than at execution time. This presents problems for the expansion of the proof tree, especially if the search space is infinite. This, nonetheless, is not an uncommon situation in partial evaluation. Uninstantiated variables can lead us to infinite solutions or infinite depth proof trees. Ways of handling this problem will be discussed in the following sections.

2.2. Suspension

The process of constructing the proof tree for a predicate may not terminate. It seems, therefore, necessary to determine the predicates that are not safe to expand during this process. These predicates have to be suspended, i.e. expansion has to be avoided. We will define a meta-predicate `unsafe/1` to be true if its predicate argument is not safe to expand, i.e. it may lead to a non-termination situation or to incorrect evaluations due to the lack of appropriate information (such as the treatment of negated predicates discussed in a latter section). If the expansion of a predicate `p` is suspended during the construction of its proof tree, `p` will become a leaf in the proof tree. Thus, the expansion of a predicate `p` is suspended if `p` is considered *unsafe*.

The problem at this point is how to decide which predicates are unsafe. To solve this problem there are several approaches. A simple and inexpensive way (during partial evaluation time) is to keep a depth counter, and suspend the expansion of a predicate when depth level h is reached. We call this procedure *depth suspension control* (DSC). This method has not yet been used in partial evaluation. In this case, the condition for expansion is checked at partial evaluation time.

The implementation of a DSC partial evaluator is shown in Fig. 2. We add two new parameters to the meta-predicate `pe/2`, which will be used to handle the DSC. The first, has to be initialized to zero The second parameter is a constant corresponding to the depth level *h*.

```
:- mode pe(+,-,+,+).
pe(Goal,[Goal],H,H):- !.

pe(Goal,[Goal],N,H):- clause(Goal,true).

pe((Goal,Others),Tree,N,H):-
                    pe(Goal,SubTree1,N,H),
                    pe(Others,SubTree2,N,H),
                    append(SubTree1,SubTree2,Tree).

pe(Goal,Tree,N,H):- clause(Goal,Body),
                    N1 is N +1,
                    pe(Body,SubTree,N1,H),
                    append([(Goal:-Body)],SubTree,Tree).
```

Figure 2. Partial evaluation with depth suspension control.

A second technique to suspend predicates at partial evaluation time is based on predicate subsumption [5]. However, the less expensive solution is to specify the unsafe predicates before the partial evaluation process. A manual method would allow the user to specify the predicates to be suspended through a process of user annotations, using a meta-predicate `unsafe/1` to be true if its argument is a predicate defined as unsafe. The implementation of meta-predicate `pe/2` in Fig. 1 only needs to add, as first definition of `pe/2`, the clause:

```
pe(Goal,[Goal]):- unsafe(Goal).
```

With this extension to `pe/2`, the unsafe predicates will be "captured" and will not be expanded. Consider, for example, a predicate `length/2` as unsafe if its first argument is an unin-

stantiated variable. In this case, it is possible to define `unsafe/1` for predicate `length` as follows.

```
unsafe(length(X,N)):- \+ ground(X).
```
[3]

Note that a predicate might be unsafe to expand under certain conditions, but it could be safe under others. For this reason, it is important to specify those conditions. The definition of unsafe predicates depends on the context in which a predicate is used. In some cases, it is possible to specify only a subset of all the conditions that make a predicate unsafe.

It should be noted that this annotation process involves the users in a programming problem. There is work on automatizing the predicate annotations as a process done before partial evaluation time [6] based on abstract interpretation of logic programs. However, this is still matter of on-going research.

3. NEW POWERFUL OPERATIONS

3.1. Freezing and Melting

An uninstantiated variable could trigger the suspension of a predicate. Given PROLOG's resolution mechanism, it is possible that a variable, not instantiated during the process of partial evaluation of a predicate, could instantiate later during the partial evaluation of other predicates. Let us consider the example shown in Fig. 3. In this example, `length/2` is considered unsafe (because of an uninstantiated variable). The partial evaluation of this predicate, therefore, is suspended.

```
foo(N):- length(X,N),
         X = ['this','is','an','example'].

unsafe(length(X,_)):- \+ ground(X).
```

Figure 3. Freezing a predicate.

During the expansion of the second predicate in the body of `foo/1`, variable X is instantiated, therefore, `length/2`, at this point, is no longer unsafe. This example proposes the necessity of temporarily suspending the expansion of an unsafe predicate, and trying its expansion at a later time.

In logic programming, these techniques are known as *freezing* and *melting* predicates. They, however, have never been used in partial evaluation. A necessary condition for freezing the expansion of a predicate is that there should be the possibility of melting it. In other words, the uninstantiated variables which made us consider the predicate as unsafe, appear in other predicates with a possibility of instantiation. Now we will give a more formal treatment of freezing and melting in partial evaluation.

Given a program P and an atom `p(X)`, let:

```
p(X)θ1 :- P1.
...                                    (3.1)
p(X)θn :- Pn.
```

be the residual program corresponding to a partial evaluation of `p(X)` wrt P. `P1`, ..., `Pn` are subgoals and θ1, ..., θn are substitutions [11]. Let `r` be an atom and `rS` a subgoal in an OR proof tree for `p(X)` wrt P.

3. `\+ ground(X)` is true if it is not the case that `X` is instantiated to a ground term.

Figure 4. Fragment of the expansion of `p(X)`.

Then, at this stage we have for a substitution ρ, the clause:

```
p(X)ρ :- rS.                                                    (3.2)
```

Let us assume that `r` becomes frozen. `S` is a subgoal, i.e. a sequence of literals. For simplicity, we will only consider the case where `S` is an atom. We could further extend `S`. Let us assume that,

```
Sσ1 :- S1.
...                                                             (3.3)
Sσm :- Sm.
```

is a residual program corresponding to the partial evaluation of `S` wrt P, with substitutions σ1, ..., σm. Graphically,

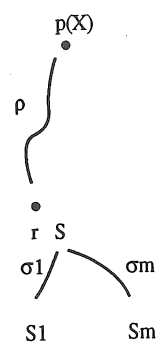


Figure 5. Expansion of `S`.

At this stage we could think of melting `r`. Notice that at the nodes `S1`, ..., `Sm`, the variables originally appearing in `S` (and also in `r`) are partially instantiated. Then, in the melting process for `r` we should further expand this partially instantiated `r`. That is, for each substitution σi, we have a residual program corresponding to the partial evaluation of `r`σi, the instantiation of `r` according to σi.

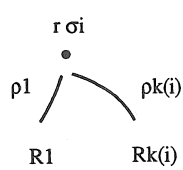


Figure 6. Expansion of `r`σi.

```
rσiρ1 :- R1.
...                                                    (3.4)
rσiρk_i :- Rk_i.
```

In order to obtain a partial evaluation of `p(X)`, i.e. the original goal, that represents this freezing-melting process at this stage, we have to combine (3.2), (3.3) and (3.4). Thus, the residual program of `p(X)` contains the new clauses:

```
p(X)ρσ1ρi :- RiS1.
...                                                    (3.5)
p(X)ρσmρi :- RiSm.
```

for i = 1, ..., k(i). To these clauses we should add the clauses corresponding to the other branches in the OR proof tree for a partial evaluation of `p(X)`. In this proof tree one of the subgoals is `S`. This subgoal was further expanded by first freezing `r` and then melting it. If `rS` is `P1` in (3.1), then the residual program for a partial evaluation of `p(X)` is given by (3.1) with the first clause replaced by (3.5).

From Fig. 5, we can see that the expansion of `S` after freezing `r` can avoid the generation of failing branches for `r`, thus producing an improvement during the execution of the residual program. Notice that the instantiations for `S` constrain the non-failing instantiations for `r` since we have to satisfy the whole subgoal `rS`.

In order to explore necessary conditions to characterize a predicate as frozen, we need to consider PROLOG's resolution strategy (such as left-to-right and top-down). Also, we need to use variable modes, either by user annotations (many PROLOG systems allow the user to declare parameter modes) or by an automatic process through static analysis [12]. If during the partial evaluation of an input program, a predicate `p` (which has been frozen) instantiates its arguments so that it is no longer considered unsafe, we say that `p` can be *melted.*

We now require an algorithm to handle frozen predicates. The predicates considered here are those predicates `p` which are unsafe to expand, but have necessary conditions to be melted. In this case, one can take one of two actions. If the predicate gets enough instantiated, one could reflect it (as explained in next section). Otherwise, expand `p` according to rules (3.5).

3.2. Reflection

Operationally, a partial evaluator translates a program in one meta-level to a lower meta-level. Sometimes, it is possible to directly execute operations which are in a higher meta-level in lower levels. This is called reflection, and will be introduced to partial evaluation as an extension.

The implementations of partial evaluators, shown in this paper, use the PROLOG unification mechanism to unify terms belonging to the predicates that are being partially evaluated. In other words, the implementations we propose are using lower meta-level features (the PROLOG interpreter) to handle predicates of an upper meta-level.

We can use this concept in a more general context. For instance, we can take an operation of a level k, send it to a level j (j < k) execute it, and return the result to level k. To implement reflection in our partial evaluator, we only have to extend the definition of predicate `pe/2` of Fig. 1, as shown in Fig. 7.

In this definition, we consider as *reflectable* a predicate if it satisfies a certain "reflectability criterion". This is defined using the predicate `reflectable/1`. It is clear that the evaluation of higher meta-level built-in predicates can be reflected, to the PROLOG meta-level, if all their input mode parameters are instantiated. The reflection of those built-in predicates whose reflection might produce side-effects, should be avoided. Finally, no other necessary conditions to define a predicate as reflectable, have been identified.

```
pe(Goal,[]):- reflectable(Goal),
              call(Goal)
```

Figure 7. Extending partial evaluation with reflection.

4. OPERATIONALIZATION

The technique of operationalization is "borrowed" from the theory of explanation based generalization (known as EBG) [14]. This is a deductive learning technique, which has never before been used in partial evaluation. In this perspective, the operationalization of a predicate is related to its reformulation in terms of other predicates, which are easier to calculate, called "operational predicates".

The main idea is to expand the proof tree of a predicate until it reaches the operational predicates, obtaining a reformulation in terms of these predicates. This reformulation is the conjunction of the leaves in the proof tree. In order to apply this technique, it is necessary to define an operational criterion capable of identifying the operational predicates during the construction of the proof tree. Obviously, suspended predicates will be defined as operational. Also, we could decide to keep some predicates in the reformulated clauses, e.g. to handle incomplete programs. For this, we use a meta-predicate `operational/1` to define the predicates in its argument as operational.

In Fig. 8, we present the implementation of a partial evaluator extended with the operational capacity (OPE). Note that the satisfaction of the operational criterion is implemented by the meta-predicate `operational/1`, which is defined in terms of the predicate `member/2`. In order to be consistent with the previous example, `son/2` and `parents/2` are considered operational.

```
:- mode pe(+,-).
pe(Goal,GoalOut):- operational(Goal),
                   copy(Goal,GoalOut)[4],
                   call(Goal),!.

pe((Goal,Goals),(Tree1,Tree2)):- pe(Goal,Tree1),
                                 pe(Goals,Tree2).

pe(Goal,Tree):- clause(Goal,Body),
                pe(Body,Tree).

operational(Goal):- member(Goal,[son(_,_),parents(_,_)]).
```

Figure 8. Implementation of an OPE.

During the construction of a proof tree in partial evaluation, it is possible to face the problem of not having enough information to decide which node to expand (in or-nodes). We can either try to expand all possible branches (with an imminent combinatorial explosion) or to abort the expansion (meaning that we operationalize the node).

5. TREATMENT OF NEGATED PREDICATES

Negation will be treated based on techniques shown in previous sections, i.e. we will consider expansion, reflection and freezing of negated predicates.

---

4. `copy(Goal,GoalOut)` is true if `GoalOut` is a copy of `Goal` with fresh variables.

5.1 Expansion of Negated Predicates

Expansion of negated predicates cannot be treated in the same way as the expansion of any other predicate, and the problem arises when the predicate is insuficiently instantiated. With the purpose of partially evaluating the negation of a predicate `p(X)`, i.e. `not p(X)`, where not stands for negation as failure, we will introduce new clauses to the residual program. These clauses have `not p(X)` in their heads. Obviously, since negation as failure is not allowed in the heads of clauses, we will replace `not p(X)` by a new atom `not_p(X)`. Expansion of a negated predicate is not equivalent to the negation of the expansion of the predicate, when uninstantiated arguments are involved. Thus, if we want an answer to the query `not p(X)` θ, we evaluate the query `not_p(X)` θ with respect to the residual program corresponding to the partial evaluation of `not p(X)`. In this section we will explain how to obtain this residual program.

As an example, let us analyze the partial evaluation of `not p(X)` in the context of Fig. 9. The partial evaluation of `p(X)` produces `p(1)` after elimination of tautologies (which will be shown later). We might be tempted to evaluate `not p(X)` wrt the partial evaluation of `p(X)`. If we do this, we obtain the non-intuitively expected answer false. In consequence, if we evaluate `not p(xo)` wrt this partial evaluation, we will always obtain the answer false. Notice that we would expect the answer *yes* for `xo=2`. An explanation for this phenomenon is that `X` was instantiated and then eliminated, losing solutions.

Intuitively, the negation of `p(X)` is true if `X` does not unify with "`1`". It is clear then, that the problem arises because variable `X` is being uninstantiated during partial evaluation.

```
p(X):- q(X),r(X).
q(1).
r(1).
```

Figure 9. Example of problems for P.E. with Negation.

Now we will describe a general procedure to construct a residual program for the partial evaluation of `not p(X)`. Let,

```
p(X)θ1 :- G1.
...
p(X)θn :- Gn.
```

be the new clauses that we replace for the definition of `p(X)` in the original program P in order to obtain the partial evaluation of the program wrt `p(X)` [11]. `G1`, ..., `Gn` are subgoals in the non-failing branches in a proof tree for `p(X)` wrt program P. This residual program will be used in the partial evaluation of `not p(X)`. Then,

```
not p(X) succeeds   ⇔   p(X) fails
                    ⇔   X ≠ Xθ1 ∧ ... ∧ X ≠ Xθn,
                        ∨
                        X = Xθ1 ∧ G1 fails
                        ∨
                        ...
                        ∨
                        X = Xθn ∧ Gn fails
```

According to this, the partial evaluation of `not p(X)` wrt P can be defined as the partial evaluation of `p(X)`, plus the following rules:

```
not_p(X) :- X =/= Xθ1, ..., X =/= Xθn.
not_p(X) :- X == Xθ1, not G1.
...
not_p(X) :- X == Xθn, not Gn.
```

The first clause corresponds to the case where the construction of the proof tree passes through a failing branch, without getting to any of the G1,..., Gn.

5.2. Freezing Negated Predicates

Freezing can be used successfully for the treatment of negated predicates. It is possible to freeze negated predicates considered unsafe if variable instantiations during partial evaluation will revert the unsafe situation, allowing predicates to melt. Here the same considerations regarding local and global freezing are valid.

5.3. Reflection in Negation

Reflection can be applied to the treatment of negated predicates, expanding the proof tree as much as possible, in order to obtain better residual programs. Let us consider the definition of a predicate p:

$$p{:-}\ L_1, L_2, \ldots, L_n.$$

where L_i ($1 \leq i \leq n$) is a negated predicate such as not q. If the proof of q produces an empty set[5], it is possible to express p as follows:

$$p{:-}\ L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n.$$

Note that this is possible since predicate q failed, and therefore, no instantiation of variables is done during the process of partial evaluation. Here, reflection is used to execute a predicate of a higher meta-level in the lower PROLOG meta-level. One of the major advantages of this technique is that it is possible to handle part of the negated predicates considered unsafe.

If the solution set of q is partitioned into $S_1 \cup S_2 \cup \ldots \cup S_n$ where S_i (S_i not empty) represents the solution i we will have:

$$\text{not } q = \text{not } S_1 \wedge \text{not } S_2 \wedge \ldots \wedge \text{not } S_n$$

allowing us to reformulate predicate p as follows:

$$p{:-}\ L_1 \wedge \ldots \wedge L_{i-1} \wedge \text{not } S_1 \wedge \ldots \wedge \text{not } S_n \wedge L_{i+1} \wedge \ldots \wedge L_n.$$

Obviously, this method works for n finite. Unfortunately, we do not yet know how to detect the unsafe cases. As an example, let us consider an example from [13] shown in Fig. 10. Even though fat and weight are finite relations, eats has a breadth-infinite solution space. A simple solution to this problem is to consider a *breadth counter control mechanism* (BCC).

```
eats(X,Y):- fat(X).
fat(X):- weight(X,Y), Y > 100.
```

Figure 10. Predicate with breadth-infinite solutions.

5. An empty solution set means that q is always false.

6. EXPERIMENTS

We wrote an interpreter of an imperative language, which we called NORMA3, to be used as a basis for our experiments. This interpreter has registers, conditional jumps, goto´s, assignments, arithmetic operations over registers, handling integer numbers, chars, strings and lists. The interpreter is written in PROLOG. We also wrote a program in NORMA3 to sort a list, based on the insertion sort algorithm.

According to Equation (1.1), the NORMA3 interpreter represents int, and the sort program is given by l. Some experiments were done, partially evaluating l wrt int, instantiating the input list to the sorting algorithm to a completely instantiated value and to a list with 3, 4 and 5 elements, obtaining residual programs ri, r3, r4 and r5, resp. These residual programs correspond to the compiled version of the imperative sorting program, i.e. we translated the sorting program from NORMA3 to PROLOG.

Three different techniques were applied to generate these residual programs, which we labeled with letters A, B and C. The first one corresponds to pure expansion (A), the second is expansion with suspension by subsumption (B), and the third is expansion, reflection, operationalization, suspension by user annotations, and tautology eliminations (C). We used a DIGITAL Decstation 3100 with Ultrix operating system and 24 MBytes of RAM, running the SICTUS PROLOG interpreter. Table 1 shows the values obtained during partial evaluation time, in CPU time units.

case	A	B	C
ri	2,331	3,316	79
r3	-	*	14,389
r4	-	*	64,054
r5	-	*	155

Table 1. Partial evaluation time.

The "-" symbol denotes that the experiment cannot be done. In the case of the previous table, this is due to infinite branches, since technique A does not have loop detectors. The "*" symbol denotes that the experiment was aborted after having generated a 6 MByte residual program. Note that the time for r5 with technique C is smaller than the ones for r3 or r4 since in the former, suspension had to be applied closer to the root, avoiding a disjunctive explosion. Table 2 shows the size of the residual programs, in number of clauses and predicates.

case	A	B	C
ri	132/422	81/244	1/1
r3	-	*	6/39
r4	-	*	24/238
r5	-	*	47/114

Table 2. Clauses/predicates in residual programs.

We can also show the execution time for the residual programs and for the sort program on top of the NORMA3 interpreter. Since the values depend on the order of the list to be sorted, we show an average time obtained doing 30 experiments each time, where the order of the elements in the list had a uniform distribution. Table 3 shows those times.

case	A	B	C	NORMA3
ri	144	128	0	50
r3	-	-	7	50
r4	-	-	35	82
r5	-	-	113.74	114.25

Table 3. Execution times.

From Tables 1 and 2 we see that the partial evaluation process with the extensions presented in this paper (C) is much more efficient than traditional partial evaluation (A, B). Also, as seen from Table 3, our extensions produce a better execution time for residual programs than the ones obtained using traditional techniques. In the case of r5 with technique C, we had to suspend the expansion quite close to the root to avoid the combinatorial explosion, and hence it was not possible to get a significant improvement in performance, as the case of ri, r3 and r4.

Note that if not enough additional information is provided to a program being partially evaluated, it will not be possible to build a deep proof tree, having to suspend the predicates near the root to avoid a disjunctive explosion due to the generation of all possible cases allowed in an algorithm. In the case of disjunctive explosion, one would be transforming the input program to a number of cases, simplifying the algorithm but loading the PROLOG interpreter with top-down search of clauses.

8. CONCLUSIONS

The extensions proposed in this paper obey the need of extending not only the functionality of current logic programming partial evaluators (with only two operations, expansion and suspension of predicates using the subsumption criterion), but also allow obtaining a better performance from the partial evaluator and the residual program.

For example, depth suspension control (DSC) increases the performance of the partial evaluator with respect to subsumption. DSC also gives the user the flexibility to obtain larger or smaller residual programs, depending on his needs, and guarantees termination of the process. The latter does not apply to the subsumption criterion.

The operations of freezing and melting predicates produce more instantiated residual programs and, therefore, more efficient programs. Reflection allows the partial evaluator to run operations from higher meta-levels into PROLOG's meta-level, and then, return their values. This is obviously a gain in partial evaluation time efficiency.

We also included operationalization of predicates, an operation "borrowed" from explanation-based generalization learning techniques, which proved to be very powerful in the construction of residual programs when it is combined with suspension, since in the reformulation one only leaves the root and the operational predicates.

The treatment of negated predicates in the input programs was also introduced. This will give the partial evaluator the capacity to analyze real programs, a feature lacking in current partial evaluators. An important development is proposed wrt expansion of non-instantiated negated predicates. Finally, a number of local optimizations were proposed in order to produce a more efficient residual program.

We have implemented a PROLOG partial evaluator which includes most of the operations here defined, and were able to process large examples of the kind described by Equation (1.1), i.e. as a program optimizer and as a universal compiler.

We are currently in the process of extending the partial evaluator to the problem of mix self-application as described by Equations (1.2) and (1.3).

ACKNOWLEDGEMENTS

This work was partially supported by the Chilean National Fund for Science and Technology (FONDECYT), grant 92-0812, and grants DIUC 91/21, 91/04E and 90/8 from the Pontificia Universidad Católica de Chile.

REFERENCES

1. S. Abramsky, C. Hankin, (eds.), 1987, Abstract Interpretation of Declarative Languages, Wiley.
2. D. Chan, M. Wallace, 1989, A treatment of negation during partial evaluation in "Meta-Programming in Logic Programming", H. Abramson and M. Rogers, eds., Chap. 16, MIT Press, London.
3. A. P. Ershov, 1978, "On the Essence of Compilation" in "Formal Description of Programming Concepts", E. J. Neuhold ed., North-Holland, pp. 391-418.
4. D. Fuller, S. Abramsky, 1988, "Mixed computation of prolog programs", New Generation Computing, Vol. 6, Springer Verlag, Tokyo.
5. D. Fuller, 1989, Partial Evaluation and Mix Computation in Logic Programming, PhD Thesis, Dept of Computing, Imperial College of Science and Technology, London, U.K.
6. D. Fuller, 1992, "Replacing the loop detection scheme in partial evaluation of logic programs", in preparation.
7. D. Fuller, S. Bocic, 1992, "Extending Partial Evaluation in Logic Programming", in Computer Science: Research and Applications, Plenum Press, NY.
8. Y. Futamura, 1971, "Partial evaluation of computation process - an approach to a compiler-compiler", *Systems, Computers, Control,* Vol. 2, No. 5, pp. 41-67.
9. N. Jones, P. Sestoft, H. Søndergaard, 1985, "An experiment in partial evaluation: The generation of a compiler generator" in "Rewriting techniques and applications", J. P. Jouannaud ed., *Lecture Notes in Computer Science*, No. 202, Springer Verlag, pp. 124-140.
10. G. Levi, G. Sardu, 1988, "Partial evaluation of metaprograms in a "Multiple Worlds" Logic Language", *New Generation Computing*, Vol. 6, Springer Verlag, Tokyo .
11. J. W. Lloyd, J. C. Shepherdson, 1989, Partial Evaluation in Logic Programming.
12. C. Mellish, 1987, "Abstract interpretation of PROLOG programs", in [1], pp. 181-198, Wiley.
13. A. Mendelzon, 1988, Logic and Databases. VIII Conference of the Chilean Computer Science Society, Santiago.
14. T.M. Mitchell, R.M. Keller and S.T. Kedar-Cabelli, 1986, "Explanation-based generalization: a unifying view" in Machine Learning Vol I.
15. L. Sterling, R. Beer, 1986, "Incremental flavor-mixing of meta-interpreters for expert system construction", Tech. Rep. TR 103-86, Center for Automation and Intelligent Systems Research, Case Western Reserve University.
16. A. Takeuchi, K. Furukawa, 1986, "Partial evaluation of prolog programs and its application to meta programming", Information Processing 86 (ed. H. Kugler), Proc. IFIP 86 Conference, North-Holland.

Evaluating the cost of the dynamic reconfiguration of a multicomputer network

J. M. García
Dpto. de Informática
Universidad de Castilla-La Mancha
Campus Universitario s/n
02071 Albacete (Spain)

J. Duato
Facultad de Informática
Universidad Politécnica
Camino de Vera s/n
46071 Valencia (Spain)

Abstract

The dynamic reconfiguration of the interconnection network is an advanced feature of some multicomputers. It allows to reduce the communication overhead, improving the performance. In a previous paper [6], we presented a reconfiguration algorithm based on a cost function. It reconfigured the network in such a way that the original topology was preserved. However, that algorithm did not take into account the cost of the reconfiguration itself. In this paper, we consider that cost. The resulting reconfiguration algorithm has been evaluated by simulating the execution of a test problem (triangularization of a sparse matrix). The results show the benefits obtained when the interconnection network is reconfigured dynamically.

Keywords: Multicomputer network, multicomputer performance, dynamic network reconfiguration, computer architecture.

1. INTRODUCTION.

Multicomputers [3] are among the most interesting architectures, meeting the high performance computing requirements in application areas like computational fluid dynamics, image processing and circuit simulation. Multicomputers rely on an interconnection network between nodes (processors) to support the message-passing mechanism.

The interconnection network plays a central role in determining the overall performance of a multicomputer [2,11]. If the network cannot provide adequate performance, nodes will frequently be forced to wait for data to arrive. Further, as technological improvements increase the computing power of nodes, additional pressure is placed on the communication network to provide a comparable improvement in performance to avoid

becoming a bottleneck. In addition to performance considerations, the interconnection network should be tolerant to failures.

A interesting solution to alleviate this problem is to make the interconnection topology reconfigurable, i.e., the network topology can change almost arbitrarily at runtime [7,10]. A network with this feature has two properties: on the one hand, it can easily match the network topology to the communication requirements of a given program, thus exploiting the *communication locality*, even when those requeriments vary over time. On the other hand, faulty nodes or links can be easily bypassed and spare nodes can be switched on. Nowadays, there are several multicomputers with this advanced feature [1,4].

In a previous paper [6], we presented a reconfiguration algorithm based on a cost function to handle the dynamic reconfiguration of the interconnection network in a multicomputer. The goal of the dynamic reconfiguration is to increase the multicomputer performance by means of minimizing the traffic of messages through the network. Therefore, the cost function evaluate the network traffic for each node, taking into account the total message traffic between a node and the other ones, weighed with the nominal distance between nodes.

This algorithm reconfigure the network in such a way that the original topology is preserved. The first results showed an important improvement for application programs with a time varying communication pattern. However, that algorithm did not take into account the cost of the reconfiguration itself.

In this paper we consider this cost, evaluating the time needed to reconfigure the network. We also show the performance improvement obtained in a complex mathematical problem, when that time is taken into account.

The rest of paper is organized as follows. Section 2 summarizes our previous work detailing the reconfiguration protocol we have implemented. In section 3 the time needed to reconfigure the network is computed. The new algorithm which includes the reconfiguration cost has been evaluated, showing in section 4 the results obtained by simulation. Finally, the main conclusions are presented in section 5.

2. MACHINE MODEL AND DYNAMIC RECONFIGURATION PROTOCOL.

This work was originated while trying to increase the communication performance of a transputer-based machine, the PARSYS SN 1000. Then, we have mainly focussed on store-and-forward networks. Also, we assume that a static scheduling is used to map processes to processors, thus preventing process migration. To handle the dynamic reconfiguration, we have set up a control in a centralized way, such as it is available in the SN 1000.

All the work has been carried out on the FDP environment [8], which permits the simulation of the behaviour of a multicomputer with a particular topology. FDP also offers a simple and efficient programming environment with a friendly user interface.

The basic idea for the dynamic reconfiguration is the following: When the traffic between a pair of nodes is intense, the reconfiguration algorithm will try to put the source node close to the destination node by exchanging the positions of either the source node and a neighbour of it or the destination node and one of its neighbours. It must be noticed that the implementation of this algorithm is distributed, each node taking into account the information recorded locally.

The dynamic reconfiguration is a suitable technique to deal with those problems whose communication pattern varies over time, especially when it is not possible to predict the variation a priori. The reconfiguration algorithm we have developed decides when it is convenient to carry out a change in the topology. The algorithm has been described in [6].

Our machine model is based on the Parsys SN1000. In this model, a master node (the system controller) is responsible for controlling the reconfiguration.

The network reconfiguration protocol works in the following way:

1).- When a node decides that it is necessary to reconfigurate the network, it sends a signal to the system controller through the control bus.

2).- Then, the system controller informs all the nodes that it is going to make a reconfiguration and therefore they should stop sending messages to each other. To minimize the reconfiguration time and relieve the cost that it implies, messages in transit are only allowed to reach the next intermediate node. After stopping messages in transit, each node sent an acknowlegement to the system controller.

3).- The node which made the request sends the reconfiguration data to the system controller to carry out the reconfiguration.

4).- The system controller modifies the interconnection network topology, adapting it to the new circumstances.

5).- Once the new configuration has been established, the system controller broadcasts this configuration to all the nodes and permits node communication again.

This protocol is easily implementable using the control bus available in the SN1000 architecture, which does not add message traffic to the network.

The centralized control could be a bottleneck for the whole system, since all the nodes must send reconfiguration requests to the system controller. To avoid it, the reconfiguration algorithm will only reconfigurate the network when it estimates that a large amount of data have to be transferred.

3. THE RECONFIGURATION COST.

As the machine is not ideal, whenever a change is made in the topology, some time is wasted, which reduces the improvement that would be obtained in an ideal machine. In this paper, we focus on the cost involved when a change in the network topology is performed.

By analysing the network reconfiguration protocol, the following times can be obtained:

a) T_a : Time spent in each node to evaluate the cost function and to decide whether a reconfiguration is needed. It measures part of the step 1 of the previous protocol.

b) T_b : Time spent to transfer the reconfiguration information from the node which requests the change to the system controller. It includes the time required to request the change as well as broadcasting the message to stop the communication between nodes, receiving the acknowledgement and sending information about the new topology to the system controller. Therefore, this time includes part of the step 1 and steps 2 and 3 of the previous protocol.

c) T_{rec} : Time needed to reconfigurate the interconnection network topology. It corresponds to step 4 of the previous protocol.

d) T_c : Time needed to broadcast the new configuration to all the nodes, allowing again communications between processors. It corresponds to the last step of the previous protocol.

Then, the cost for making a change in the network topology is given by the following expression:

$$T_{change} = T_a + T_b + T_{rec} + T_c \qquad (1)$$

Let us compute an upper bound for this expression. T_a is the time required to compute the cost function. We estimate a value of 100 μsec for it.

The communications between the system controller and the nodes are performed through the control bus, requiring less than 100 μsec. With regard to the time needed to stop message traffic, we are going to compute the time required to transfer a message between adjacent nodes. In [12] the following formula is used:

$$T_{comm} = \alpha + \beta * L \qquad (2)$$

where L is the length of the message (in bytes) and α and β are machine dependent constants. Usually, long messages are split into fixed size packets. For the transputer [9], we have obtained for T_{comm} a value ranging from 500 μsec to 1 ms, giving for T_b an estimated value ranging from 600 μsec to 1.1 ms.

The other constant factor, T_c, also measures communications through the control bus. Again, we estimate a value equal to 100 μsec.

Finally, we have to evaluate the time needed to perform a reconfiguration in the interconnection network. In [5] the following formula to compute the reconfiguration time is proposed:

$$T_{rec} = \gamma + \delta * N \qquad (3)$$

where γ is the start-up time for the reconfiguration, δ is the required time per modified link and N is the number of modified links in each reconfiguration. Since changes are local, the number of links which are modified in each change is always the same. When two nodes exchange their positions, the number of modified links is equal to 6. Using the values proposed in [5] for γ (100 μsec) and δ (100 μsec/link), the time needed to perform a network reconfiguration is equal to:

$$T_{change} = 800 + 100 + 100 * 6 = 1{,}500\ \mu sec = 1{,}5\ ms \qquad (4)$$

for short messages. For long messages, $T_{change} = 2$ ms.

4. EVALUATION OF THE ALGORITHM.

To evaluate the performance of the reconfiguration algorithm in a complex case, we have chosen the triangularization of a sparse matrix based on Givens rotations. This algorithm is very suitable for parallel machines because of its inherent parallelism. This algorithm has been chosen because the communication pattern will vary over time, making it very suitable for dynamic reconfiguration. A detailed explanation of the algorithm as it has been simulated under FDP can be found in [7].

The results we show measure the total message traffic in the network. In the case of dynamic reconfiguration, we can take into account the reconfiguration cost. The reconfiguration cost is approximately equal to the time required to transfer three messages between adjacent nodes in the worst case (short messages). Therefore, it is important to limit the number of changes.

In figure 1 we show the total traffic of messages through the network for a 600x300 sparse matrix with an average of two non-zero elements per row. For comparison purposes, we show the results obtained with three static topologies (ring, 2-D mesh and hypercube) and the results obtained when network reconfiguration is allowed, both with (real reconfiguration) and without (ideal reconfiguration) the reconfiguration cost.

In the case of reconfiguration, the basic topology is a hypercube. The values obtained for the network traffic vary as a function of different parameters from the reconfiguration algorithm [6].

As can be seen in the figure, the reduction in the network traffic is drastic with respect to the static hypercube. However, the reduction is smaller when compared with the ring topology, because this is the optimal topology when the matrix is full. It must be noticed that small matrices are quickly filled.

These results have been confirmed with many tests with matrices of different sizes [7]. The results obtained are very good, showing that the larger the matrix, the better results are achieved.

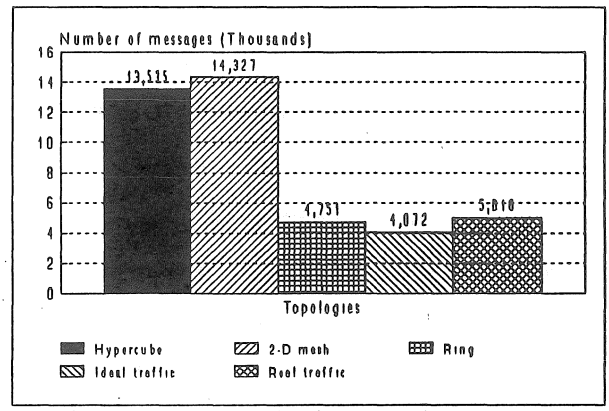


Fig. 1. Message traffic in the network for a 600x300 matrix

These results confirm the suitability of the dynamic reconfiguration as a means of achieving a reduction in message traffic, therefore improving the performance of multicomputers.

5. CONCLUSIONS.

In this paper we have presented the dynamic reconfiguration of the interconnection network as a valid alternative to solve the main problem associated to multicomputers: the communication bottleneck. By means of using a dynamic topology, the principle of locality in communications is exploited, leading to a substantial improvement in network latency [2].

The main contribution of this paper is the evaluation of the time needed for reconfiguring the topology according to the proposed protocol. Once this time has been computed we have shown how the use of the network reconfiguration leads to a reduction in message traffic.

REFERENCES

[1] Adamo, J. and Bonello, C. "Tenor++: A dynamic configurer for Supernode machines". *Lecture Notes in Computer Science* No. 457, pp. 640-651, Springer-Verlag, 1990.

[2] Agarwal, A. "Limits on interconnection network performance". *IEEE Trans. on Parallel and Distributed Systems*, Vol. 2, No. 4, pp. 392-412, October 1991.

[3] Athas, W. C. and Seitz, C. L. "Multicomputers: Message-passing concurrent computers". *IEEE Computer,* Vol. 21, No. 8, pp. 9-24, August 1988.

[4] Bauch, A.; Braam, R. and Maehle, E. "DAMP: A dynamic reconfigurate multiprocessor system with a distributed switching network". *Proc. 2nd European Distributed Memory Computing Conference,* Munich, April 1991.

[5] Desprez, F. and Tourancheau, B. "Evaluation des performances de la machine T.node". La lettre du Transputer, LIB Besacon, France, No. 7, 1990.

[6] García, J.M. and Duato, J. "An algorithm for dynamic reconfiguration of a multicomputer network". *Proc. Third IEEE Symposium on Parallel and Distributed Processing,* Dallas, December 1-5, 1991.

[7] García Carrasco, José M. "*Desarrollo de Herramientas para una Programación Eficiente de las Redes de Transputers: Estudio de la Reconfiguración Dinámica de la Red de Interconexión*". PhD thesis. Universidad Politécnica de Valencia. December, 1991.

[8] García, J.M. and Duato, J. "An Advanced Environment for Programming Transputer Networks with Dynamic Reconfiguration". *Int. Conf. on Parallel Computing and Transputer Applications,* Barcelona, September 1992.

[9] Inmos Corporation. *"The Transputer Databook".* Inmos Ltd., England, 1989.

[10] Nicol, D.A. "Reconfigurate transputer processor architectures", in T.J. Fountain and M.J. Shute (Ed), *Multiprocessor Computer Architectures,* North-Holland, 1990.

[11] Reed, D.A. and Fujimoto, R.M. "*Multicomputer Networks: Message-based parallel processing".* The Mit Press. London, England, 1987.

[12] Zhang, X. "System effects of interprocessor communication latency in multicomputers". *IEEE Micro,* pp. 12-15 and 52-55, April 1991.

The Proto-Architecture of WOLF, a Dataflow Supercomputer

A. Garcia Neto* C. A. Ruggiero †

Grupo de Instrumentação e Informática
Departamento de Física e Ciência dos Materiais
Universidade de São Paulo, Brasil

Abstract

This article presents the proto-architectecture studied to define the characteristics of the Wolf processor, a proposal for the implementation of a high-performance supercomputer based on the dataflow paradigm. In order to place this proposal in the context of research in dataflow architecture, a survey of the main efforts in this area carried out in Europe, North America and Japan is done.

Keywords: Parallel Architectures, Dataflow, MIMD Implementations.

1 Introduction

The Wolf architecture is loosely based of the Manchester Multi-ring Dataflow Machine(MMDM) [26, 27, 59], incorporating many of the lessons learned since the implementation of that prototype ten years ago and the independent studies and modelling done elsewhere [20, 21].

The Wolf architecture is based on the following premises:

- Dataflow machines address elegantly and efficiently some difficult problems ocurring in other parallel machines, such as process allocation, load distribution, linear speed-up, memory contention and access latency. Dataflow machines are also easy to program, can expose and exploit parallelism without the programmer's help and has a well-defined mechanism to avoid the exposition of excessive parallelism [42, 51].
- The von Neumann architecture can process algorithms of low parallelism more efficiently than dataflow architectures [18, 30].
- SIMD architectures are more efficient than dataflow architectures to exploit parallelism derived from regularity in data structures [31].
- The RISC approach allows the implementation of highly efficient sequential processors [41, 57].
- Dataflow architectures are profligate in memory usage [18, 19].
- The circuits needed to implement Matching Memories using content-addressable components, common to most dynamic dataflow architectures, are complex and wasteful of silicon.
- Parallelism should be extracted out of algorithms without burdening the programmer with architectural details of the parallel machine.

2 Dataflow Architectures

Most dataflow implementations are based either on the the *ring* and the *network* organization, typified by the now classical designs of Manchester and of the MIT. This section places the development of the Wolf architecture in the context of previous work in dataflow.

2.1 Research in Europe

2.1.1 Manchester

The Manchester Dataflow Research Group has proposed the ring-based architecture [24, 25, 26, 27, 59] of figure 1.

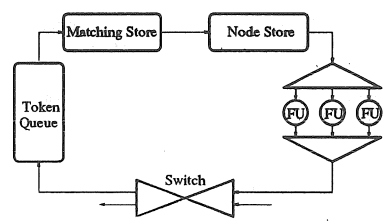


Figure 1: The Manchester Architecture

The MMDM is a collection of rings connected by a routing network. Within each ring there are units to smoothe the traffic (token queue), to implement the firing rule (the matching unit), to store the graph (the node store) and to execute the dataflow instructions (the functional units). The MMDM imple-

*alvaro@uspfsc.ifqsc.usp.ansp.br
†toto@uspfsc.ifqsc.usp.ansp.br

ments a dynamic tagged model. The programming language of the prototype is SISAL.

2.1.2 DTN

The DTN Dataflow Computer is a commercially available graphics-oriented workstation with 32 PE static dataflow engines by the Dutch company Dataflow Technology Netherland [58]. The workstation uses the ImPP [54] developed by NEC, coupled to a VME Unix system and a graphics subsystem with four 64 MOPS systolic arrays.

2.2 Research in North America

2.2.1 The VIM Machine

The Dennis group at MIT pioneered many of the basic dataflow concepts [16, 15, 17]. Research has continued from 1968 to date, currently aimed at the construction of a 1 GFlop static dataflow machine for large numerical computation. The VIM machine, shown in figure 2, is a collection of routing networks, cell blocks (CB), functional units (FU) and array memories (AM).

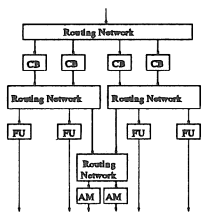


Figure 2: The VIM architecture

The cell blocks store the data dependency graphs and the data structures needed to implement the dataflow firing rule and termination acknowledgements, and execute simple operations such as duplicating tokens. The functional units consume the data scheduled by the cell block. Structured data are stored in the array memories. The routing network tolerates latency. The programming model for VIM is the VIMVAL [28], and extension of the VAL language [14, 10] that treats functions as first class objects, that can be passed and returned as parameters.

2.2.2 The MIT-Tagged Machine

This research was begun at Irvine in 1975 and is continued at the MIT by Arvind's Grup, evolving into the proposal of the MTTDA (the MIT Tagged-Token Dataflow) [3, 5, 6], with the architecture of figure 3:

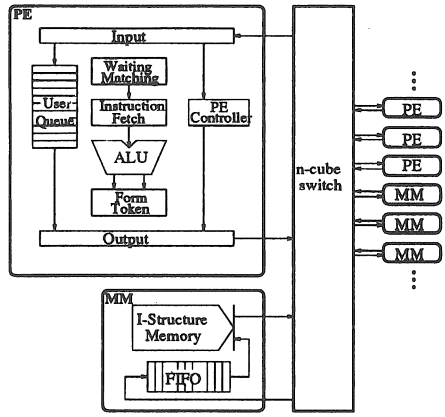


Figure 3: The MTTDA architecture

The MTTDA is an dynamic dataflow asynchronous machine with 64 PEs connect using a n-cube communication network and *I-structure nodes* that uses the tagging concept to enable more than one instantiation of a routine to share the same data dependency graph. The I-structure node stores static data and supports out-of-order reads and writes. A 256 board machine rated at 1 GIPS is under construction. The programming language for the machine is the dataflow language Id. The machine is aimed at both symbolic and numerical computations.

2.2.3 The Monsoon Machine

The Monsoon architecture [40] is a multithreaded one-address machine incorporating the Explicit Token Store (ETS) concept [13]. A Monsoon machine comprises a number of pipelined processors and I-structure memories connected by a routing network, as shown in figure 4.

The ETS concept requires that the frame pointer be local to a processor, and thus, activation names are restricted to the PE to where they were allocated, and execute entirely there. Because of the ETS, the firing rule is implemented as a indexed direct fetch to the frame store, which controls the arrival of partners.

The pool of tokens waiting to be processed is distributed, implemented as a two queues within the PE. The higher priority queue the *System Queue* FIFO, the lower priority queue is the *User Queue* stack. A PE ha a pipeline with the following stages: Instruction Fetcher, Effective Address Calculator, Presence Bits Operator (communication with the Presence Bits Memory), Frame Operator (communicating with the Frame Memory), ALU (three stages, including New Tag Calculator and communicating

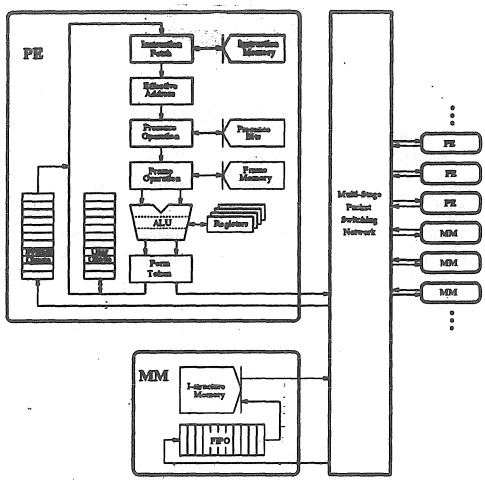


Figure 4: The Monsoon architecture

with a Register Bank), and Token Formator. A path has been provided to recirculate one of the result tokens back to the pipeline.

A 4 MIPS single PE prototype has been operational since 1988, executing a kernel SO written in Id. A joint MIT/Caltech/Motorola collaboration is developing a 10 MIPS, 10 MFLOPS single board PE with VME interface using 8 CMOS arrays with 10,000 gates [4]. The inter-PE routing network is implemented with PaRC 4x4 switching ICs [32]. A single PE Unix workstation running the Id dataflow language [38] is expected to be launched by Motorola in 1992. A system with 8 PEs, 8 Memory Modules, 8 Switch Modules and 4 Unix I/O Modules is also expected to be operational by then.

2.2.4 Other architectures

DDM1: The Data Driven Machine was designed at Borroughs in 1975. It is a colourless dynamic dataflow design, using FIFO queues to distinguish between different instantiations of graph reutilization. It has a topology of an octary tree connecting PEs containing an Agenda Queue, an Atomic Memory, an Atomic Processor and a Switching module.

TDDP: A Distributed Data Processor designed at Texas Instruments in 1976 to investigate the feasibilitiy of a static high performance dataflow design implementing a model that does not require confirmation of execution [11, 12]. A prototype using TTL logic was operational by 1978 using four PEs connected as a ring, using ADA as a programming language.

Epsilon-1, developed at Sandia National Laboratories as an evolution ot the DFAM project, was a test-bed dataflow processor that attained sustained uniprocessor performance comparable to contemporary mini-supercomputers [23, 22]. It uses *direct match*, a technique that enables single-cycle access to the matching unit implementing the dataflow firing rule, and uses *repeat-on-input*, a technique that minimizes the overhead associate with limited fan-out of dataflow instructions in most implementations.

Epsilon-2 is a successor of Epsilon-1 that fully supports the dynamic dataflow model. It is a variable-grain machine using a hybrid scheduling mechanism that enables both sequential and dataflow execution. The Epsilon-2 architecture comprises a number of PEs connected by a routing network, as shown in figure 5.

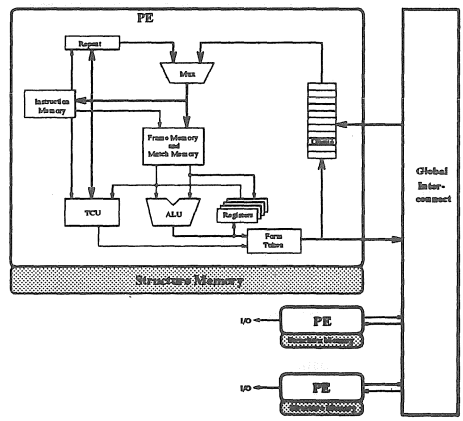


Figure 5: The Epsilon-2 architecture

The PE architecture draws from the Epsilon-1 and Monsoon PE architectures, and executes code compiled in Id, Sisal and Fortran. The PEs have support for I-structures, and load balancing is implemented by biasing the routing network as to direct allocation to the least loaded PE, in a scheme reminiscent of EM-4.

2.3 Research in Japan

2.3.1 SIGMA-1

Developed at the Eletrotechnical Laboratory (ETL/MITI), the SIGMA-1 project studies the feasibility of constructing a dynamic dataflow computer for numerical computation of realistic size [29, 49]. This project started in 1982 and was completed by 1988. The computer has over 128 synchronous PEs and sructure memories grouped in clusters and connected by a dual-level hierarchical network, as shown in figure 6.

The hardware is a synchronous microprogrammed CISC processor based on a 10,000-gate semi-custom

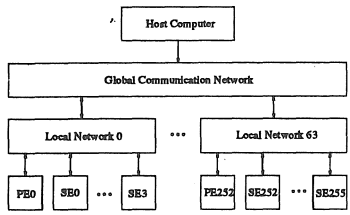


Figure 6: The Sigma-1 architecture

LSI, with 27 chips per board. Theoretical peak performance is 423 MFlops, 640 MIPS for a 128 node processor. Performances over 170 MFlops have been reported [45, 51]. Instructions have 40 bits for tagged data, and can generate results to up to three other instructions. SIGMA-1 supports DFC-2, a C-like language with single-assignment restrictions.

2.3.2 EM-4

The fourth generation of symbolic processing machines of ETL, EM-4 is a variable grain dynamic dataflow machine using the concept of strongly connected arc [34, 51]. A single chip RISC dataflow processor, called EMC-R, was built in 1988 with 50,000 gate semi-custom LSI technology [44]. A EMC-R encompasses the basic dataflow function: packet switching, input buffering, instruction fetch, operand matching, and instruction execution. An 80 node prototype is operational since 1990, as shown in figure 7.

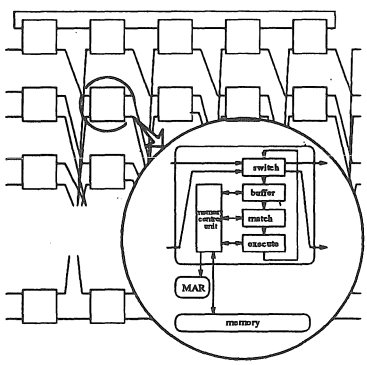


Figure 7: The EM-4 architecture

The theoretical peak performance of the prototype is 1.1 GIPS for a 80 node processor, 14.6 GIPS for a 1024 node [51].

2.3.3 Other Projects in Japan

Many other dataflow machines have been proposed, built or simulated such as:

NEDIPS developed at Nippon Telegraph and Telephone Corporation (NTT) in the late 1970s, is a hardwired static dataflow architecture dedicated to image processing with an extension to support multiple tokens per arc.

ImPP, the Image Pipelined Processor developed at NTT in the early 1980s is an implementation of NEDIPS in a single chip integrated circuit, intended to be used as a building block to larger systems. The ImPP is commercially available since 1985 [58].

EDDY, developed at NTT, is an array of 16 nodes with *nearest neighbour* connectivity. Each node has two Z8000 microprocessors. It uses VALID as a programming language. The project started in 1980, the prototype ran in 1983 [53] and was reported finished in 1986.

TopStar of Tokio University is a Z80 microprocessor-based prototype with 16 PEs, exploiting dataflow at the inter-procedure level [48] aimed at the recognition of printed Chinese character patterns. The project started in 1978 and finished by 1982, and is followed by the TOPSTAR-II and, subsequently, the PIE projects. THe porgramming languages used are Lisp and Prolog.

DDDP of Oki Electric Industry Co is a four processor, fixed-logic small scale prototype for numerical applications, started in 1980 and finished in 1982 [33].

DFM at NTT. A feasibility study for a dataflow processor for symbolic manipulation and list-processing, with lazy cons strategy [1, 2, 39]. Project started in 1982, a two-processor prototype ran in 1985. The high level language is VALID. A CMOS LSI version, DFM-II was started in 1985.

PIM-D of Oki Electric Industry Co. is a 16 processor, small-scale prototype for numerical applications, started in 1982 and operational in 1984 Project. It uses PROLOG as a programming model.

EM-3 of ETL is a Lisp machine, with 16 PEs based on the Motorola 68000 microprocessor. The project started in 1982 and the prototype is operational since 1985 [60]. It uses EMLISP, a single-assignement LISP subset.

DFNDR-1 at Gunma University, is a four processor, microprocessor-based (6809), small scale prototype for performance evaluation [47]. It used static dataflow, and the project was reported finished by 1986.

TIP, the *Template-based Image Processor* is a commercial static dataflow integrated circuit developed by NEC for image processing [54], available since the early 1980s.

Q-v1 a joint development of Osaka University, Sharp, Mitsubishi, Sanyo and Matsushita, directed to signal processing and image processing [56]. It is an 8 PE prototype using a five chip set based on the Manchester design, implementing a dynamic tagged

dataflow model [35] and using the elastic pipeline approach [36].

3 The Wolf Architecture

3.1 Known Problems

Many lessons have been learned during the implementation of the MMDM:

- The circular shape of the pipeline increases latency time and causes traffic fluctuation.
- The number of stages of the MMDM pipeline is larger than required, and hampers performance.
- The asynchronous implementation of the modules causes delays in the pipeline and makes project debbuging very complex.
- The fixed data word size of 32 bits is inadequately small for some applications and excessively large for others
- The granularity of the MMDM instructions seems to be finer than it should be.
- The fixed granularity of the MMDM hampers research about optimal granularity level in dataflow processors.
- There is redundancy in several token fields, unnecessarily increasing the matching space.

3.2 A Pathological Example

The data dependency graph of figure 8, a typical code for parameter-passing in function calls of SISAL programs compiled for the MMDM [37] shows some of the disadvantages of the dataflow approach [7, 8, 9].

Figure 8: Pathological code.

In this code, a sequential ordering of instructions is enforced by the data dependency. The von Neumann architectures, due to their implicit sequentiality, are much more efficient in these cases. In a typical dataflow processor, no information about the scheduling of instruction `Op2` can be made at the time of the triggering of the `Op1` instruction. However, it is easy to see that the firing rule that schedules `Op1` has also scheduled, sequentially, `Op2, Op3` and `Op4`. Since a typical dataflow processor cannot see this implicit scheduling, the token with the result from `Op1` will have to go through an entire pipeline round for each of one of the instructions `Op2, Op3` and `Op4`.

This inneficiency is made worse by the fact that no instruction following `Op1` needs any further input data. The single output from the previous instruction is their sole input. In the MMDM terminology, these unary tokens are called BYPASSes, but are still sent to the matching unit, where a dummy firing mechanism (which always yields a match) is used. Simulations show that up to 60% of all executed instructions can have unary inputs [55].

Another problem detected in this situation is the creation of "short-lived" tokens, such as the intermediary result of `Op1` needed to trigger `Op4`. In a typical dataflow processor, this intermediary result is stored in the matching unit and almost immediately retrieved. This pointless exercise, coupled with the traffic of unary instructions, making the matching unit the slowest and most critical unit of the pipeline.

3.3 Working Description

The proto-Wolf Supercomputer consists of the interconnected units shown in figure 9. In the following working description, the nomenclature from the MMDM has been retained where possible.

A token represents a piece of data on an arc of the data-dependency graph being executed. Tokens are inserted into the machine via the input datapath of the Collecting Network. Answers are sent to the host computer through the output datapath of the Distributing Network.

From the Collecting Network, tokens pass through the Token Queue to the Matching Unit. The Matching Unit is responsible for matching operands so that both operands of a dyadic operator are made available to the subsequent units. It combines the output from the Data Memory to forms a Group Package, a pair of normal tokens directed to the same node in the data-dependency graph and with the same label. The Group Package is distributed through the Distributing Network to one an available parallel Node Store. If no Node Store is available, the Group Package is grabbed by the Overflow Unit for latter distribution. The Node Store holds the details of the node which the pair of operands have reached. It proceeds an Executable Package containing all the information required to execute the node, and forwards it to its associated Functional Unit. The results produced by the Functional Unit can be either *free* or *chained*. Free results are passed to the Collecting Network, and thence round the ring. Chained results remain in the Functional Unit internal registers, to be, at some stage, used as operands to subsequent

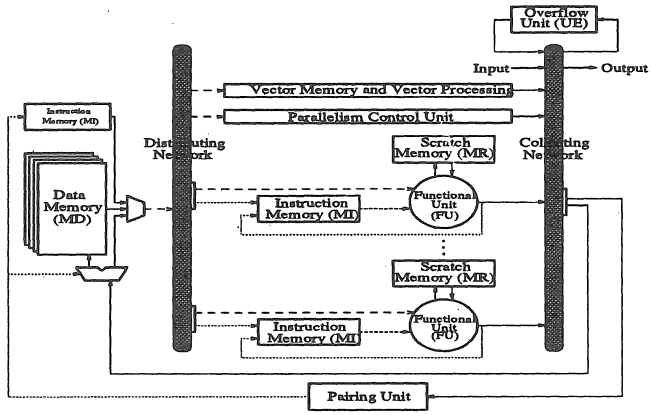


Figure 9: The Wolf Dataflow Architecture

operations. If the Collecting Network cannot forward the received tokens, the Overflow Unit stores them to form a pool of available data for latter distribution

One of the details held by the Node Store is wether a given node is part of a chain or not. When an Executable Package belonging to a chained node is produced, the Node Store begin an extra fetch cycle, using the destination of the recently produced package as address. This spontaneous fetch cycle progresses in parallel, while the Functional Unit consumes the aforementioned Executable Package. Thus, when the consumption is finished, a new Executable Package is produced from the Node Store ready to be executed, using the results of the previous package as input.

Connected to the Collection and Distributing Networks are one or more Structure Store units, used to store data. The Structure Store allocates space on request, and supports a reference count mechanism, automatically reclaiming the space if the reference count is ever reduced to zero. The write and reads of a location may be issued in any order, the store deferring within itself requests for access to a value which is not yet present. It is, however, an error to write to a location which is already occupied. Besides reading from a location, which is non-destructive, it is possible to extract from location, leaving it empty. Messages to the Structure Store are constructed by Structure Store orders executed in the Functional Units.

Tokens are inserted into the machine via the input datapath of the Collecting Network. Answers are sent to the host computer through the output datapath of the Distributing Network.

From the Collecting Network, tokens pass through the Token Queue to the Matching Unit. The Matching Unit is responsible for matching operands, where necessary, so that both operands of a dyadic operator are made available to the subsequent units. It combines the output from the Data Memory to form a Group Package, a pair of tokens directed to the same node in the data-dependency graph and with the same label. The Group Package is distributed through the Distributing Network to any available parallel Node Store. The Node Store holds the details of the node which the pair of operands have reached. It proceeds an Executable Package containing all the information required to execute the node, and forwards it to its associated Functional Unit. The results produced by the Functional Unit can be either *free* or *chained*. Free results are passed to the Collecting Network, and thence round the ring. Chained results remain in the Functional Unit internal registers, to be at some stage used as operands to subsequent operations.

The Node Store holds whether a given node is part of a chain or not. When an Executable Package belonging to a chained node is produced, the NS starts autonomously an extra fetch cycle, using the destination of the recently produced package as address. This spontaneous fetch cycle progresses in parallel, while the Functional Unit consumes the aforementioned Executable Package. Thus, when the consumption is finished, a new Executable Package is produced from the Node Store ready to be executed, using the results of the previous package as input.

3.4 Architecture Characteristics

3.4.1 Variable Granularity

The main characteristic of the Wolf processor is its ability to execute code as either a dataflow processor or as a von Neumann processor. This concept has already explored elsewhere [20, 30] in the context of variable granularity. The Wolf architecture,

shown in figure 9, enables a stream of sequential instructions to be generated and presented to a Functional Unit. In this units, this stream of functions will successively operate upon the accumulator register, where the result of the previously executed instruction is still stored.

3.4.2 Locality

The stream of instructions is generated whenever a flag bit is planted by the compiler. The detection of these sequential chains in data dependency graphs is common practice in many situations, and does not pose special difficulties. Conversely, "short-lived" results can be easyly detected, and special LOAD and STORE instructions planted to store this result in a scratch memory. The resulting tokens can be directed directly to the Instruction Memory, where new streams of instructions are triggered. Thus, many disjoint groups of sequential instructions can be executed between the pair Functional Unit-Node Store before a reference to the global matching mechanism.

With this enforced locality, the issue of load distribution arises. In typical dataflow processors, the spread of fine grain instructions is sufficient to the ensure an efficient, transparent and almost cost-free balancing mechanism. With enforced locality, some processors might become overload while others idle. The Wolf architecture avoids this situation by providing separate paths for matching tokens and for unary tokens.

The Connecting Network that gathers the output from the Functional Units has four exit paths: One directs data outside the machine, two direct data to the matching mechanism and one sends data directly to the Distributing Network. This last path is provided to enable an overloaded Functional Unit to offload part of its traffic to some other Functional Unit. Each Functional Unit communicates to the Distributing Network its level of activity, and the network is capable of directing the traffic to the least busy Functional Unit. This approach was successfully employed in the SIGMA-1 dataflow computer [50, 52]

3.4.3 Parallelism Control and Vector Processing

The processing elements between the Distributing and Collecting Networks do not have to be homogeneous. One known characteristic of dataflow designs is its ability to expose more parallelism than required, flooding the machine with more (potentially) parallel actions than the number of available processing elements [43]. One solution for this problem is to limit the number of different instantiations of the data dependency graphs, limiting the creation of new *Activation Names* [42, 55] This is done by the Parallelism Control Unit, which also performs the recycling of these activation names by identifying instantiations that have already finished.

At this same level stands the Vector Processing Unit, which exploits parallelism in data structure regularity. In Wolf, these regular data structures do not circulate in the ring: Instead, pointers are processed, and, when the actual operation is due, an action is requested to the Vector Processing Unit. This aproach was suggested by Sargeant for the Structure Store, but never implemented [46].

3.4.4 Matching Unit and Data Memory

Since the operations remain constrained to a single Functional Unit, the enforced locality dispenses with large quantities of activation names. The implementation of the Parallelism Control Unit at the same hierarchical level of the Functional Units allows for effective Activation Name recycling, and the Vector Processing Unit dispenses with most indexes in tokens.

These factors concur for a large decrease in the size of the matching space, which becomes compatible to the sizes of current virtual memories. The Data Memory, can now be managed using the same proven techniques. In Wolf, it is a standard paged virtual memory, addressed by the Matching Unit.

The Collecting Network has two separate paths for the matching mechanism: one takes tokens to the Data Memory, the other takes addresses to the Matching Unit. The Matching Unit uses this address to verify if a pair to this token is waiting in the Data Memory. If there isn't a waiting pair, it generates a write signal which stores the incoming data in the required address. If a pair is already waiting, the Matching Unit generates a read signal and forwards the address. The Data Memory is responsible for joining the incoming data with its own stored data. Its output is a group of tokens with the same tag and destination address.

4 Implementation Details and Aims

- Variable word length, from 32 to 96 bits.
- Reduced token size (as compared to the MMDM): 26, 56, 152 and 164 bits.
- Variable granularity: Fine grain, stream of instructions and thick grain.
- Functional Units implemented using RISC processors with scratch memory.
- Matching Unit independent of Data Memory. Random addressable Data Memory.

- Matching of up to four tokens.
- Instruction Memories directly coupled to the Functional Units, capable of generating instruction streams using *prefetch*.
- Separate data paths for bypass and matching tokens.

The units of Wolf exchange information solely by tokens, which are subdivided in *fields*. The widths of the inter-module busses is one of the critical points of most dataflow designs. The size of the token fields used in Wolf, which determine the width of the busses, are shown in table 1.

Field	Name	Function	Size
$\mathcal{A}$	Activation	Identification of activation	10
$\mathcal{C}$	CodOp	Operation Code	6
$\mathcal{D}$	Data	Data	32
$\mathcal{E}$	Matching	Number of matching tokens	2
$\mathcal{N}$	Node	Address of instruction in program	12
$\mathcal{T}$	Type	Data type	4

Table 1: Fields of the tokens

References

[1] M. Amamiya and R. Hasegawa. Dataflow Computing and Eager and Lazy Evaluation. *J. New Generation Computers*, 2(8):105–129, 1984.

[2] M. Amamiya, R. Hasegawa, O. Nakamura, and H. Mikami. Implementation and Evaluation of a List Processing-Oriented Dataflow Machine. In *Proc. 13th Annual Int. Symp. Computer Architecture*, 10–19. IEEE/ACM, 1986.

[3] Arvind and D. Culler. Dataflow Architectures. CSG Memo 294, LCS, MIT, Cambridge, Mass., USA, 1986.

[4] Arvind, M. L. Dertouzos, R. S. Nikhil, and G. M. Papadopoulos. PROJECT DATAFLOW, a Parallel Computing System Based on the Monsoon Architecture and the Id Programming Language. Technical Report CSG Memo 285, LCS, MIT, 1988.

[5] Arvind, V. Kathail, and K. Pingaley. A Dataflow Architecture With Tagged Tokens. CSG Memo 174, LCS, MIT, Cambridge, Mass., USA, Sep 1980.

[6] Arvind and R. S. Nikhil. A Dataflow Approach to General-Purpose Parallel Computing. CSG Memo 303, Lab. Computer Science, MIT, Cambridge, Mass., USA, Jul 1989.

[7] A. P. W. Böhm and J. Sargeant. Efficient Dataflow Code Generation for SISAL. Internal Report 2nd issue, DCS, Univ. Manchester, England, Sep 1985.

[8] A. P. W. Böhm and J. Sargeant. Efficient Dataflow Code Generation for SISAL. In M. Feilmeier, G. Joubert, and U.Schendel, ed., *Parallel Computing 85*, 339–344. North Holland Publishing Co, Jun 1986.

[9] A. P. W. Böhm and J. Sargeant. Code Optimisations for Tagged-Token Dataflow Machines. *IEEE Trans. Computers*, 38(1):4–14, Jan 1989.

[10] J. D. Brock. Operational Semantics of a Data Flow Language. Technical Report TM–120, LCS, MIT, Cambridge, Mass., USA.

[11] M. Corsish. The TI Dataflow Architectures: The Power of Concurrency for Avionics. In *Proc. 3rd Conf. on Digital Avionics Systems*, 19–25, Forth Worth, TX, USA, Nov 1979. IEEE.

[12] M. Corsish, D. W. Wogan, and J. C. Jensen. The Texas Instruments Distributed Processor. In *Proc. Louisiana Computer Exposition*, 189–193, Lafayette, LA, USA, Mar 1979.

[13] D. E. Culler and G. M. Papadopoulos. The Explicit Token Store. *J. Paralle and Distributed Computing*, 10(4), Jan 1991.

[14] J. B. Dennis. First Version of a Dataflow Procedure Language. In *Proc. Colloque sur la Programacion. v.19 (Lecture Notes in Computer Science)*, 362–376. Springer-Verlag, Berlin, 1984.

[15] J. B. Dennis. Modelling the Weather with a Dataflow Supercomputer. *IEEE Computer*, 3(7):592–603, 1984.

[16] J. B. Dennis. Models of Data Flow Computation. In *Proc. 1984 IEEE CompCon*. IEEE Computer Society, 1984.

[17] J. B. Dennis. Data Flow Computation. In M. Broy, editor, *NATO ASI Series, v.F14, Control Flow and Data Flow: Concepts of Distributed Programming*, 346–397. Springer-Verlag, Berlin, 1985.

[18] J. F. Foley. Manchester Dataflow Machine: Benchmark Test Evaluation Report. Internal Report UMCS–89–11–1, DCS, Univ. Manchester, England, Nov 1989.

[19] D. D. Gajski, D. A. Pádua, D. J. Kuck, and R. H. Kuhn. A Second Opinion on Data Flow Machines and Languages. *IEEE Computer*, 15(2):58 – 69, Feb 1982.

[20] D. Ghosal and L. W. Bhuyan. Analytical and Architectural Modifications of a Data Flow Computer. *Computer Architecture News*, 15(2):81–89, Feb 1987.

[21] D. Ghosal and L. W. Bhuyan. Performance Evaluation of a Dataflow Computer. *Trans. Computers*, 39(5):615–627, May 1990.

[22] V. G. Grafe and J. E. Hoch. Implementation of the Epsilon Dataflow Processor. In *Proc. 23rd Hawaii Int. Conf. on System Sciences*, 1990.

[23] V. G. Grafe and J. E. Hoch. The Epsilon-2 Multiprocessor System. *J. Parallel and Distributed Computing*, 10(4):309–318, Dec 1990.

[24] J. R. Gurd et al. Fine-Grain Parallel Computing: The Dataflow Approach. Dataflow Group internal document in preparation, DCS, Univ Manchester, England.

[25] J. R. Gurd, C. C. Kirkham, and W. Böhm. *The Manchester Dataflow Computing System*, v. 1 of *Special Topics in Supercomputing*. North-Holland, Jan 1987.

[26] J. R. Gurd, I. Watson, and J. R. W. Glauert. A Multilayered Dataflow Computer Architecture. Internal report, DCS, Univ. Manchester, Mar 1980.

[27] J. R. Gurd, I. Watson, and C. C. Kirkham. The Manchester Prototype Dataflow Computer. *Comm. ACM*, 28(1), Jan 1985.

[28] J. Herath, Y. Yamaguchi, N. Saito, and T. Yuba. Dataflow computing models, languages, and machines for intelligent computations. *IEEE Trans. Software Engineering*, 14(12):1805–1928, Dec 1988.

[29] K. Hiraki, T. Shimada, and K. Nishida. A Hardware Design of the SIGMA-1 — A Data Flow Computer for Scientific Computations. In *Proc. Int. Conf. on Parallel Processing*, 524–531. IEEE, 1984.

[30] R. A. Iannucci. A Dataflow/von Neumann Hybrid Architecture. Technical Report TR–418, LCS, MIT, Cambridge, Mass., USA. 1988.

[31] R. N. Ibbet. *The Architecure of High Performance Computers*. The MacMillan Press, London, first edition, 1982.

[32] C. F. Joerg. Design and Implementation of a Packet Switched Routing Chip. Master's thesis, Dept. Electrical Eng. and Computer Science, MIT, Cambridge, Mass., USA, 1988.

[33] M. Kishi, H. Yasuhara, and Y. Kawamura. DDDP: A Distributed Data Driven Processor. In *Proc. 10th Annual Int. Symp. Computer Architecture*, 236–242. IEEE, 1983.

[34] Y. Kodama, S. Sakai, and Y. Yamaguchi. A Prototype of a Highly Parallel Dataflow Machine EM-4 and Its Preliminary Evaluation. ETL Research Memorandum, Eletrotechnical Laboratory, 1990.

[35] S. Komori, K. Shima, S. Miyata, T. Okamoto, et al. The data-driven microprocessor. *IEEE Micro*, 9(3):45–60, Jun 1989.

[36] S. Komori, H. Takata, T. Tamura, F. Asai, T. Ohno, O. Tomisawa, T. Yamasaki, K. Shima, et al. An Elastic Pipeline Mechanism by Self-Timed Circuits. *J. Solid-State Circuits*, 23(1):111–117, Feb 1988.

[37] J. R. McGraw. SISAL: Streams and Iteration in a Single Assignement Language, Language Reference Manual, Version 1.2. Technical Report M–146, Lawrence Livermore National Laboratory, Mar 1985.

[38] R. S. Nikhil. Id (version 88.0) Reference Manual. Technical Report CSG Memo 284, LCS, MIT, Mar 1988.

[39] S. Ono, N. Takahashi, and M. Amamiya. Optimized Demand-Driven Evaluation of Functional Programs on a Dataflow Machine. In *Proc. Int. Conf. on Parallel Processing*, 421–428, 1986.

[40] G. M. Papadopoulos. Implementation of a General Purpose Dataflow Multiprocessor. PhD Thesis, Technical Report TR 432, LCS, MIT, Cambridge, Mass., USA, Sep 1988.

[41] D. A. Patterson and C. A. Sequin. A VLSI RISC. *IEEE Computer*, 15(9):8–21, Sep 1982.

[42] C. A. Ruggiero. Throttle Mechanisms for the Manchester Dataflow Machine. Technical Report Series UMCS-87-8-1, DCS, Univ. Manchester, England, Jul 1987.

[43] C. A. Ruggiero and J. Sargeant. Control of Parallelism in the Manchester Dataflow Machine. In *Proc. ACM Int. Symp. Functional Programming Languages and Computer Architecture*, 1 – 17, Oct 1987.

[44] S. Sakai, Y. Yamaguchi, K. Hiraki, Y. Kodama, and T. Yuba. An Architecture of a Dataflow Single Chip Processor. In *Proc. 16th Annual Int. Symp. Computer Architecture*, 46–53, Jerusalem, Israel, Jun 1989. IEEE/ACM.

[45] M. Sakurai. Data-flow demon; MITI masters 640 MIPS. *Electronic Engineering Times*, (481):45–46, Apr 1988.

[46] J. Sargeant. Efficient Stored Data Structures for Dataflow Computing. Technical Report Series UMCS-85-8-2, DCS, Univ. Manchester, England, Aug 1985.

[47] M. Sowa and T. Murata. A Data FLow Computer Architecture with Program and Data Memories. *IEEE Transacitions on Computers*, C-31(9):820–824, 1982.

[48] T. Suzuki, K. Kurihara, H. Tanaka, and T. Moto-oka. Procedure. Level Data Flow Processing on Dynamic Structure Multimicroprocessors. *J. Information Processing*, 1(5):11–16, 1982.

[49] T. Shimada and K.Hiraki and K. Nishida. An Architecture of a Data Flow Machine and Its Evaluation. In *Proc. COMPCON 84 (Spring)*, 486–490. IEEE, 1984.

[50] T. Shimada and others. Evaluation of a Prototype Data Flow Processor of the SIGMA-1 for Scientific Computations. In *Proc. 13th Int. Symp. Computer Architecture*, 226–234. IEEE/ACM, Jun 1986.

[51] T. Shimada T. Yuba et al. Dataflow Computer Development in Japan. *ACM*, 140–147, 1990.

[52] T. Shimada T. Yuba, K. Hiraki et al. The SIGMA-1 Dataflow Computer. Internal report, Eletrotechnical Laboratory, MITI, Japan.

[53] N. Takahashi and M. Amamiya. A Data Flow Processor Array System — Design and Analysis . In *Proc. IEEE 10th. Annual Int. Symp. Computer Architecture*, 243–250. IEEE, 1983.

[54] T. Temma, S. Hasegawa, and S. Hanaki. Dataflow Processor for Image Processing. *Proc. on Mini and Microcomputers*, 5(3):52–56, 1980.

[55] Y. M. Teo. Concurrency Control in the Multi-Ring Manchester Dataflow Machine. Master's thesis, Univ. Manchester, England, Sep 1989.

[56] H Terada. H Nisikawa nd K. Asada, S. Matsumoto, S. Miyata, S. Komori, and K. Shima. VLSI Design of a One-Chip Data-Driven Processor: Q-v1. In *Proc. Fall Joint Computer Conf.*. ACM/IEEE, 1987.

[57] F. J. Valente. Estudo de arquiteturas risc. Master's thesis, DFCM, IFQSC, Universidade de São Paulo, 1991.

[58] A. H. Veen and R. van den Born. The RC Compiler for the DTN Dataflow Computer. *J. Parallel and Distributed Computing*, 10(4):319–332, Dec 1990.

[59] I. Watson and J. R. Gurd. A Practical Dataflow Computer. *IEEE Computer*, 15(2), Feb 1982.

[60] Y. Yamagushi, K. Toda, J. Herath, and T. Yuba. EM-3: A LISP-Based Data-Driven Machine. In *Proc. Int. Conf. on Fifth Generation Computer Systems*, 524–532. ICOT, 1984.

Otimizações Importantes para Paralelismo OU em Prolog sobre Máquinas com Memória Distribuída

Cláudio Geyer, Jacques Briat, Michel Favre, J. Chassin de Kergommeaux*

Projet CMaP, Laboratoire de Génie Informatique - IMAG
BP 53 X,
38041 Grenoble Cedex
França

Resumo

Este artigo descreve novas otimizações adotadas no modelo Opera ([3]) de execução do paralelismo OU multi-seqüencial em Prolog, para máquinas com memória distribuída. Uma técnica especial para a compilação das variáveis condicionais, baseada na máquina abstrata de Warren, é descrita, reduzindo o número de ligações profundas. Um novo método para a implementação eficiente da instalação de tarefas, baseado na cópia de contextos, é apresentado. A cópia é executada em paralelo ao trabalho da WAM, exigindo a inclusão do valor da ligação na pilha de registro de ligações. Esse método consome menos memória e diminue a complexidade da cópia incremental. Uma técnica é proposta para a implementação do predicado de corte (*cut*), mantendo-se a semântica original de Prolog, sendo mais simples e consumindo menos memória que a de [1]. Uma arquitetura hierárquica do sistema é proposta para as configurações com grande número de processadores (> 100) da máquina Supernode, sobre a qual foi implantado o protótipo Opera.

Palavras-chaves: paralelismo OU, Prolog, memória distribuída, otimizações.

1. Introdução

Este artigo descreve algumas otimizações importantes adotadas no modelo Opera de execução do paralelismo OU em Prolog, para máquinas com memória distribuída. Essas técnicas não foram abordadas em artigos anteriores sobre o modelo Opera ([3]), por motivos diversos como falta de espaço e, principalmente, uma evolução após a redação daqueles artigos.

O modelo Opera, para a execução do paralelismo OU em Prolog, foi concebido dentro dos estudos desenvolvidos para a obtenção de aumentos de desempenho da linguagem Prolog (equipe Flop, em Grenoble). Tais estudos estão centrados principalmente na exploração do paralelismo. A maior simplicidade do paralelismo OU foi a principal razão para a sua escolha como objetivo inicial, em detrimento do paralelismo E ([7]).

* Atualmente na Universidade Federal do RGS, Instituto de Informática.
Av. Bento Gonçalves, 9500 - Bloco IV - Bairro Agronomia
Caixa Postal 15064 - 91501 - Porto Alegre/RS - Brasil
N. Fax: (55) (051) 336-5576 - E-mail: geyer@inf.ufrgs.br

Desde o início, optou-se pelo paralelismo implícito, sem alteração de Prolog, o que permite a execução direta de aplicações existentes, desenvolvidas em Prolog. Essa abordagem também mantem as qualidades de linguagem declarativa de Prolog, permitindo uma programação paralela de mesma complexidade que a seqüencial ([7]).

A arquitetura alvo de Opera é a classe das máquinas com memória distribuída, devido a essas máquinas serem novidade e apresentarem um potencial de maior quantidade de processadores.

Considerando-se, como hipótese, que as máquinas de memória distribuída não são (ainda) próprias para comunicação muito freqüente, o modelo Opera foi concebido como um modelo OU multi-seqüencial. Tais modelos apresentam maior granularidade que os modelos OU de cláusula, pois uma tarefa é definida como uma resolvante completa (cláusula e continuação). Como modelo multi-seqüencial, Opera tem as seguintes características principais:

- Opera é constituído de N máquinas abstratas seqüenciais Prolog (MAP), onde N é função dos recursos em memória e processador da máquina alvo;
- cada MAP executa uma interpretação procedural clássica, criando nós OU e fazendo retrocessos quando ela atinge uma folha da árvore OU;
- no início da execução, todas as MAP estão inativas, uma só recebendo a resolvante inicial e tornando-se ativa;
- cada MAP inativa torna-se ativa pela aquisição de tarefas suspensas armazenadas nos nós OU das máquinas ativas;
- uma MAP ativa torna-se inativa quando, chegando a uma folha da árvore OU, ela não possuir alternativas suspensas nos seus nós OU;
- a execução termina quando todas as máquinas tornam-se inativas.

A gestão dos contextos múltiplos é feita por duplicação ([2]), e não por compartilhamento de dados como na maioria dos modelos encontrados na literatura. Tal característica implica, quando da migração de uma tarefa de máquina ativa para inativa, na transferência (cópia) do contexto relativo ao nó OU ao qual pertence a tarefa. Essa cópia é justificada, na ausência de uma memória comum, pelo relativamente alto custo de um acesso não-local em máquinas com memória distribuída, e pelas características dos acessos de uma MAP: freqüentes e a células pequenas (por exemplo, palavra de memória). Além disto, a operação de cópia pode usufruir do recurso de comunicação por DMA (em paralelo ao cálculo), existente nas máquinas paralelas em geral. Essa cópia de contexto é efetuada de modo incremental, isto é, as seções do contexto a ser transferido, já existentes na máquina inativa, não são transferidas ([2]).

Em Opera, adotou-se a compilação como método de implementação de Prolog. Uma implementação seqüencial ineficaz, por exemplo por interpretação, poderia conduzir a ganhos artificiais no desempenho do sistema paralelo, pois o custo da comunicação seria relativamente dimuído pelo real aumento do tempo de execução seqüencial. A MAP de Opera é baseada na máquina abstrata de Warren (WAM [6]), a mais utilizada em implementações de Prolog.

Um protótipo desse modelo foi implementado sobre uma máquina Supernode, baseada em Transputers, com memória distribuída ([3, 4]). Tal protótipo apresentou bons ganhos de desempenho, como por exemplo 14 para o programa queens(10) sobre 16 processadores. No

entanto, durante o desenvolvimento e avaliação do protótipo, diversas otimizações foram concebidas. Algumas dessas otimizações, decritas nesse artigo, são:

- conceito de variáveis condicionais aplicado à compilação;
- novo método de instalação de uma tarefa, baseado em cópia de contexto;
- implementação do operador de corte;
- arquitetura de processos para escalonamento e protocolo de comunicação para configurações do Supernode com mais de 100 processadores.

Uma análise preliminar das duas primeiras otimizações indicam a possibilidade de maiores ganhos de desempenho. A terceira, ou operador de corte, é necessária para a execução de muitos programas Prolog reais, e a quarta é exigida pela característica hierárquica das configurações maiores do Supernode.

2. Variáveis Condicionais

Esta seção descreve uma proposição de compilação para determinadas variáveis WAM, aqui denominadas condicionais ([4]). O objetivo é a redução do custo extra gerado por essas variáveis na execução da operação de cópia de contextos.

A compilação de Prolog em um programa WAM mapeia as variáveis Prolog em diferentes tipos de variáveis WAM. As variáveis WAM são classificadas em ([6]):

- locais: não são referenciadas em estruturas, sendo alocada para cada uma somente uma célula no registro de ativação (um por cláusula) correspondente da pilha local;
- globais: ao menos uma das referências é feita em uma estrutura, sendo que, em caso de modo de escrita da estrutura, uma célula é alocada na pilha global, além da célula da pilha local.

A alocação e a inicialização de uma célula de variável na pilha local são realizadas por intruções distintas, podendo ocorrer entre as duas a criação de um nó OU. Tal fenômeno não ocorre com as células da pilha global.

Uma **ligação condicional** de uma variável WAM é aquela que é separada da alocação da variável por um ou mais nós OU. Quando de uma cópia de contexto, uma ligação condicional deve ser desfeita pela WAM inativa, caso a tarefa exportada corresponda a um dos nós existentes entre a alocação da variável e a ligação. Tal operação exigiria a transferência de toda a pilha de registro de ligações e a suspensão momentânea do trabalho da WAM ativa. Portanto, uma cópia simples de contexto implica uma forte sincronização com o trabalho normal da WAM, e a transferência e o exame de registros de ligações não-necessárias à WAM inativa.

Para evitar esses dois problemas, o modelo Opera introduziu uma pilha de variáveis e a datação (profundidade dos nós OU). A cada ligação condicional, o valor e a data de ligação são registradas nessa pilha adicional numa célula alocada com a inicialização da variável. A cópia de contexto transfere essa pilha, e a máquina inativa desfaz as ligações examinando a data de cada uma, prescindindo da pilha de registro de ligações ([3]).

O uso da cópia incremental introduz um problema de coerência com relação às variáveis locais condicionais. Uma variável condicional é copiada para uma máquina inativa em estado ainda não-inicializado, caso a cópia seja realizada entre a alocação e a inicialização da variável (o nó OU da alternativa transferida foi criado entre a alocação e a inicialização). Caso a alternativa da máquina inativa falhar antes da inicialização e essa máquina importar uma alternativa posterior à inicialização, a variável em questão restará não-inicializada, pois a mesma não será novamente copiada. Uma variável não-inicializada implica erros de execução, caso seja usada em determinadas instruções da WAM.

A solução para evitar esse problema é simples. As variáveis condicionais locais, cuja primeira ocorrência encontra-se após o primeiro objetivo (*goal*), deverão ser inicializadas logo após as instruções geradas para a cabeça da cláusula. Para tal, em Opera, uma nova instrução *init_perm(X)* é gerada para cada uma dessas variáveis, e a primeira ocorrência é compilada como *value* (leitura da variável sem inicialização), e não como *variable*. Essa inicialização "avançada" implica um sobre-custo em relação ao procedimento normal da WAM, pois em caso de falha em algum objetivo da cláusula um maior número de incializações se tornará inútil.

A pilha de variáveis e a datação (PVD) introduzem um sobre-custo constante nas operações de alocação, inicialização e ligação das variáveis. Com o intuito de reduzir esse sobre-custo, todas as ligações incondicionais devem ser efetuadas de modo **superficial**, isto é, sem uso do mecanismo PVD. Determinar à compilação a característica incondicional condicional de uma ligação é um problema complexo, o que implicaria uma implementação totalmente dinâmica para essa otimização.

Por outro lado, é possível classificar as variáveis em **incondicionais**, aquelas que certamente terão ligações incondicionais, e **condicionais**, aquelas que eventualmente terão uma ligação condicional. A diferenciação na compilação desses dois tipos de variáveis é simples, pois elas já são normalmente compiladas de modo diferente na WAM.

As variáveis incondicionais são as variáveis locais aparecendo na cabeça da cláusula. Elas serão certamente ligadas a um argumento da chamada do próprio predicado, podendo usufruir do mecanismo de ligação superficial. As variáveis condicionais são as outras variáveis locais e todas as globais. Para as variáveis condicionais locais, torna-se obrigatório a alocação e inicialização da célula na pilha de variáveis quando da inicialização pela *init_perm*. A cada ligação, verifica-se o carácter incondicional condicional e, no primeiro caso, efetua-se uma ligação superficial.

As variáveis globais cuja primeira ocorrência é no corpo da cláusula exigem uma inicialização com alocação de célula na pilha de variáveis, efetuadas ao mesmo tempo quando dessa ocorrência. Quanto às variáveis globais que ocorrem na cabeça da cláusula, propõe-se uma compilação diferenciada, pois existe a possibilidade de que elas não sejam ligadas por essas ocorrências. A última ocorrência, anterior à primeira chamada (até o primeiro objetivo inclusive), é compilada com versões especiais das respectivas instruções WAM. Essas versões alocam e inicializam uma célula na pilha de variáveis, caso a variável ainda esteja livre e caso ele seja criada (modo gravação para *unify_variable*, *unify_local_value* e *unify_value*).

3. Instalação de Tarefas

A instalação de uma tarefa consiste de três etapas:

- preparação dos dados a enviar, com atualização dos estado da WAM exportadora (pilha de nós OU);
- envio e recepção das seções (não-comuns) das pilhas necessárias à execução da tarefa;
- restauração do estado da WAM importadora (registradores e ligações condicionais).

O método proposto anteriormente em [3], baseado na pilha de variáveis e na datação, apresenta alguns inconvenientes quando acoplado à cópia incremental e à transferência de alternativas pertencentes à mais de um nó OU.

A cópia incremental elimina a transferência das seções comuns às duas WAM, mas não exclue a restauração das ligações condicionais sobre as variáveis dessa seção. Em outros termos, as ligações efetuadas pela WAM importadora, após o nó OU mais recente entre os com alternativas transferidas, devem ser desfeitas, assim como as ligações efetuadas pela WAM exportadora, antes do mesmo nó, devem ser instaladas na WAM importadora.

Duas alternativas gerais são possíveis para efetuar-se a instalação das ligações da WAM exportadora:

- envio completo da pilha de variáveis, isto é, inclusive a seção comum. Essa alternativa desfaz também as ligações da WAM importadora, mas ela inclue ligações já existentes na WAM importadora (redundância de dados);
- determinação dessas ligações pela WAM exportadora (por exemplo, percorrendo a seção comum da pilha de variáveis e analisando a data de ligação), com conseqüente envio a WAM importadora.

A transferência de alternativas pertencentes a vários nós OU exige a cópia de determinadas seções da pilha de registro de ligações. Essas seções, situadas entre aqueles nós, serão necessárias quando de retrocessos (locais) da WAM importadora. No entanto, essas seções e as seções transferidas da pilha de variáveis contem informações redundantes.

Assim, com o intuito de reduzir os sobre-custos acima do método PVD, propõe-se um outro método para a instalação de tarefas, baseado em uma pilha de variáveis mais simples e em uma modificação da pilha de registro de ligações (PVV [4]). A pilha de variáveis contem somente o valor da ligação, excluindo-se a data de ligação. A pilha de registro das ligações contem, além do endereço da variável, o valor da ligação.

O princípio geral desse método é a transferência exclusiva das ligações válidas à WAM importadora. Essa validade é determinada pelo mais recente nó OU entre os transferidos.

A instalação é realizada com as seguintes etapas:

- os nós OU são transferidos, enquanto a WAM importadora desfaz as ligações efetuadas por ela própria após o nó OU comum entre as duas WAM;

- o nó OU comum e o nó OU transferido mais recente são utilizados para a determinação das seções não comuns das pilhas;
- a seção não-comum da pilha de registro de ligações é transferida;
- as seções não-comuns das pilhas global e local são transferidas, enquanto a WAM importadora instala na pilha de variáveis as ligações existentes na seção não-comum da pilha de registro de ligações.

Esse método apresenta uma sincronização de condição, entre a recepção e o trabalho de instalação efetuados pela WAM importadora: a recepção da pilha de registro de ligações somente pode começar após o fim da desligação das ligações anteriores ao nó OU comum. O tempo de retardo corresponde à diferença entre o fim da recepção dos nós OU e o fim da desligação, podendo então ser nulo.

Uma etapa adicional, para a inicialização da seção não-comum da pilha de variáveis, é necessária caso a mesma não seja estática (total, no início da execução).

Uma comparação preliminar entre esse método e o PVD (com percurso da seção comum da pilha de variáveis para determinação das ligações a transferir) resulta em:

I1 - diferença na quantidade de dados a transferir: 2 vezes (2 células) o número de variáveis condicionais da seção não-comum (PVD - pilha de variáveis) contra o número de ligações efetuadas durante a seção não-comum (PVV - pilha de registro de ligações contem uma célula a mais);

I2 - diferença no cálculo (iterações sobre as pilhas): número de variáveis condicionais da seção comum (PVD) contra o número de ligações condicionais efetuadas durante a seção não-comum (PVV);

I3 - espaço (alocação da segunda célula de ligação condicional: data ou valor na pilha de registro de ligações): possibilidade de alocação sem que a ligação ocorra (PVD) contra alocação somente em caso de ocorrência de ligação (PVV).

Conclusões definitivas quanto aos dois primeiros itens (I1 e I2) exigem medidas experimentais sobre um número representativo de programas Prolog, no sentido de avaliar-se as quantidades médias de variáveis condicionais não-comuns e comuns, e de ligações condicionais efetuadas durante a seção não-comum. Na falta dessas medidas, o método PVV parece mais indicado devido ao fator 2 no primeiro item (I1) do método PVD, ao fato de que numa mesma seção o número de ligações é no máximo igual ao número de variáveis e ao item I3.

4. Operador de Corte (cut)

O método proposto para a implementação do operador de corte (*cut*) no modelo Opera é um método de marcação dos nós OU, dependentes de um corte em potencial, com conseqüente inibição do paralelismo ([4]). Um método eficaz deveria evitar a penalização de cláusulas sem cortes, o que é difícil devido a falhas ocorridas antes da execução do corte e a cláusulas singulares (sem nó OU associado) contendo cortes.

No método Opera, o nó OU atual, no início da execução da primeira alternativa (cláusula ou alternativa de disjunção) com corte, é marcado através de um novo registrador BFREE. Ele é o nó

OU mais recente não inibido. As alternativas, que contem operadores de corte e que não são associadas a um nó OU, são contadas até a criação de um nó OU, utilizando-se de um registrador adicional NCC. A exportação de um nó OU é inibida se ele é posterior ao nó BFREE.

A marcação é efetuada pelas instruções de retrocesso (*try*, *retry* e *trust*) e por uma nova instrução. Tal instrução é gerada no início de uma cláusula, caso a mesma contenha um ou mais operadores de corte. A retirada da marca é realizada pelas instruções de retrocesso e por instrução de corte gerada para o último (versão especial da instrução WAM de corte) corte da cláusula.

Um registrador adicional, SNCC, é usado para o salvamento do contador de cortes (NCC) quando da marcação, esse valor sendo necessário no momento de retrocesso ao primeiro nó inibido.

5. Arquitetura hierárquica

Por limitações de espaço, essa otimização será descrita de forma muito resumida. O protótipo inicial de Opera contem um escalonador centralizado, o qual se adapta perfeitamente à arquitetura da menor configuração da máquina Supernode, denominada Tnode. Nesse protótipo, cada nó contem, além do processo WAM, um processo espião e um processo exportador. O processo espião informa o escalonador da carga da WAM associada (ativa). O escalonador é alocado em um processador central (de controle) com capacidade de comunicação 1 * n por via de comunicação e controle. O processo exportador está sempre à espera de um pedido de exportação por parte do escalonador, sendo responsável pela transferência de contextos. Quando uma WAM inativa solicita uma tarefa ao escalonador, esse analisa o estado das cargas das WAM ativas e escolhe uma entre elas para ser a WAM exportadora, enviando então às duas WAM os respectivos pedidos de importação e exportação.

No entanto, configurações maiores do Supernode, denominadas Meganode, são constituídas de vários Tnode, interligados através de um Transputer de controle adicional, podendo comunicar-se com os Transputer de controle dos Tnode de modo 1 *n por outras vias de controle.

Para tais máquinas, propõe-se uma nova organização do escalonador, hierárquica, onde um processo semelhante (EC) ao do protótipo original é alocado no processador de controle central para fins de escalonamento inter-Tnode, e processos de escalonamento local (EL) são alocados nos Transputer de controle de cada Tnode.

Os processos EL alimentam o EC com informações sobre as suas cargas, basicamente a maior carga de uma máquina ativa local. Tal informação será usada na escolha do Tnode exportador em caso de requisição de importação por um EL sem carga. O processo EC, ao contrário da versão anterior sobre um Tnode, não mantem o estado das seções comuns, necessário à cópia incremental ([4]), pois considera-se que instalações inter-Tnode serão raras, não compensando o custo de controle de seções comuns.

Cada processo EL atende as necessidades de escalonamento (seleção das máquinas exportadora-importadora), como anteriormente, isto é, dando prioridade aos pedidos das máquinas locais. No entanto, em caso de não haver máquina local com capacidade de exportação,

e havendo requisições de importação, essas são repassadas ao processo EC. Além disto, o processo EL deverá atender eventuais requisições de exportação (se possível), encaminhadas pelo EC, a partir de requisições de importação, produzidas por máquinas de outros Tnode, e repassadas ao EC pelos outros EL.

Note-se por exemplo, que um processo EL poderá, ao receber uma autorização de importação inter-Tnode, recusá-la pois alguma de suas máquinas ativas conseguiram, após a decisão de requisitar tal importação, atingir um estado de carga mínimo para exportação. Tais situações aumentam a complexidade dos processos EL, com relação ao escalonador do protótipo sobre um Tnode simples.

6. Conclusão

O protótipo Opera inicial, baseado no modelo pilha de variáveis e datação, apresentou bons ganhos de desempenho, mesmo sem conter diversas otimizações previstas no modelo, como a cópia incremental. A implementação da mesma verificou-se complexa, e com alta sincronização e alguma redundância. Nesse artigo, propos-se uma evolução do modelo, baseada principalmente na cópia de ligações condicionais válidas à WAM importadora e na redução do número de variáveis condicionais. A modificação correspondente do protótipo é necessária, com o intuito de validar essa evolução.

Referências:

[1] Ali, K.A.M. A Method for Implementing Cut in Parallel Execution of Prolog. In Proceedings of 4th Symposium on Logic Programming, San Francisco, pp.449-456, 1987.

[2] Ali, K.A.M. and Karlsson, R. The Muse Or-Parallel Prolog Model and its Performance. In Proceedings of North-American Conference on Logic Programming 90, Austin, pp.757-776, 1990.

[3] Briat, J., Favre, M., Geyer, C., Chassin, J.K. Scheduling of Or-Parallel Prolog on a Scalable, Reconfigurable, Distributed-Memory Multiprocessor. In Proceedings of PARLE'91, 1991.

[4] Geyer, C. Une Contribution à l'Étude du Parallélisme OU en Prolog sur des Machines sans Mémoire Commune. Tese de doutorado. Université Poseph Fourier, Grenoble, 1991.

[5] Sohma, Y et al. A New Parallel Inference Mechanism Based on Sequential Processing. Tecnhical Memorandum No. TM-0131, ICOT, 1985.

[6] Warren, D.H.D. An Abstract Prolog Instruction Set. Tecnhical Note No.309, Artificial Intelligence Center, SRI International, 1983.

[7] Warren, D.H.D. The SRI Model for Or-Parallel Execution of Prolog - Abstract Design and Implementation Issues. In Proceedings of 4th Symposium on Logic Programming, San Francisco, pp.92-102, 1987.

Aplicação de Paralelismo na Síntese de Imagens Através do Método da Radiosidade

Gladimir V. Guimarães Baranoski
Leonardo de Carvalho Garcia

Informatics Institute - CPGCC - UFRGS - Caixa Postal 15064 - Porto Alegre - Brasil
FAX: (051) 336-5576 - E-mail: gladimir@inf.ufrgs.br garcia@inf.ufrgs.br

Abstract

This paper presents the first application of the Cooperative Processes Software (CPS) on radiosity methods using a parallel strategy based on software rather than hardware. The application is described and the preliminary results as well as the projections regarding future efforts in this area are pointed out.

1. Introdução

O emprego de imagens geradas por computador vem recebendo um destaque significativo nos últimos anos. Estas imagens têm importantes aplicações envolvendo a simulação de fenômenos naturais e a produção de sequências de animação com finalidades educacionais ou de entretenimento. Os esforços nesta área da computação gráfica visam aumentar o grau de realismo das imagens e desenvolver processos de síntese mais acurados do ponto de vista da aplicação de leis físicas.

O método da radiosidade [1] destaca-se entre os métodos de síntese de imagens por apresentar um tratamento mais acurado da simulação das reflexões de luz. Entretanto, este método apresenta um tempo de processamento significativo. Com a finalidade de reduzir este tempo vários pesquisadores estão desenvolvendo estudos propondo o emprego de arquiteturas paralelas em aplicações envolvendo este método. Estes estudos propõem alternativas paralelas baseadas em "hardware" [2-3].

Este trabalho propõe a aplicação de paralelismo na síntese de imagens através do método da radiosidade, utilizando uma ferramenta de "software" conhecida como *"Cooperative Processes Software" (CPS)* [4-6]. Será descrita a aplicação proposta de paralelismo nas diferentes etapas do processo de síntese e serão apresentados e analisados os resultados já obtidos.

2. Fundamentos do Método da Radiosidade

O método da radiosidade foi introduzido por Goral *et al.* [1] e Nishita e Nakamae [7]. É um método orientado para a síntese de imagens por computador, baseando-se em princípios de conservação de energia e transferência de energia calorífica por radiação. Estes princípios são utilizados há bastante tempo em engenharia e foram adaptados de forma a modelar a interação de energia radiante luminosa (luz) entre superfícies refletoras.

Os principais aspectos que diferenciam o modelo de iluminação apresentado pelo método da radiosidade dos modelos anteriores ao seu surgimento são os seguintes:

- modelar a reflexão de luz entre objetos e/ou superfícies refletoras difusas (as quais compõem a maioria dos ambientes reais);
- incluir efeitos de mistura de cores ("color blending") causados pelas reflexões difusas;
- empregar fontes de luz de área finita e
- ser independente da posição do observador.

Este método consiste basicamente no cálculo das *radiosidades* dos N setores, ou "patches", que compõem as superfícies de um ambiente fechado. A radiosidade de um "patch" é a soma da luz emitida e da luz refletida por ele (1). Para calcular as radiosidades é necessário especificar tanto as relações geométricas entre os "patches" que compõem o ambiente fechado, os *fatores de forma*, quanto os aspectos de emissão luminosa, *reflectividade* e *emissividade*, associados a cada "patch".

$$B_j = E_j + \rho_j \sum_{i=1}^{N} B_i F_{ij} \tag{1}$$

onde:

B_j = radiosidade da superfície j, representa a taxa total de energia radiante luminosa que deixa a superfície j em termos de energia por unidade de tempo e por unidade de área;

E_j = emissividade da superfície j, representa a taxa de energia radiante luminosa que deixa a superfície j por unidade de tempo e por unidade de área;

ρ_j = reflectividade da superfície j, representa a fração de energia radiante luminosa incidente, que é refletida pela superfície j de volta para o ambiente;

B_i = radiosidade da superfície i;

F_{ij} = fator de forma, representa a fração de energia radiante luminosa que deixa a superfície i e atinge a superfície j.

Os fatores de forma representam o ponto crítico do método da radiosidade, já que além de influenciarem diretamente na exatidão das radiosidades, o seu cálculo corresponde a uma parcela bastante significativa do tempo total gasto na síntese das imagens. Existem diversos métodos de cálculo de fatores de forma. Neste trabalho optou-se pela utilização do *Método do Diferencial Paramétrico (MDP)* [8]. Este método trata convenientemente superfícies curvas e apresenta um elevado grau de exatidão no cálculo dos fatores de forma.

Uma vez calculados os fatores de forma e arbitrados os valores de emissividade e reflectividade de cada "patch" é montado o sistema de N equações lineares (2) que caracteriza o balanço de energia do ambiente fechado. A fim de se obter a radiosidade completa de cada "patch", é necessário montar e resolver este sistema de equações para cada banda de interesse do sistema de cores que se deseja empregar. Neste trabalho utilizou-se o método conhecido como *"LU Decomposition"* [9] na resolução dos sistemas de equações.

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & -\rho_1 F_{1N} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & -\rho_2 F_{2N} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ -\rho_N F_{N1} & -\rho_N F_{N2} & 1-\rho_N F_{NN} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \cdot \\ \cdot \\ \cdot \\ B_N \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \cdot \\ \cdot \\ \cdot \\ E_N \end{bmatrix} \tag{2}$$

Na etapa seguinte, emprega-se um algoritmo de remoção de elementos ocultos para determinar os elementos do ambiente visíveis ao observador. Neste trabalho, optou-se pelo algoritmo de *"ray-casting"* [10]. Na etapa final, as radiosidades obtidas são normalizadas e interpoladas de forma a permitir a determinação da intensidade de cada "pixel" da imagem. Estas duas últimas etapas constituem o processo de *"rendering"* propriamente dito das imagens.

3. Descrição da Ferramenta

O *CPS* é uma ferramenta que reúne um conjunto de facilidades para a programação, depuração e execução de aplicações paralelas e foi originalmente concebida para a utilização no multiprocessador *ACP II* [4-5]. Criado

no Fermi National Accelerator Laboratory, o projeto *ACP II/CPS* destina-se basicamente ao processamento dos dados obtidos em experiências de física experimental de altas energias [4], as quais apresentam uma demanda computacional elevada. O *CPS* destina-se à implementação de paralelismo de alta granularidade. Os objetos que executam em paralelo são definidos por processos de usuário com capacidade de comunicação e sincronização entre si. Estes processos podem ser executados paralelamente quando cada processo é alocado individualmente a uma máquina ou concorrentemente quando um conjunto de processos é alocado a uma mesma máquina.

O *CPS* foi desenvolvido para plataformas compatíveis com *UNIX* atualmente incluindo "drivers" para os sistemas SUN, Apollo e Silicon Graphics (estações de trabalho) e VAX ("mainframe" executando *VMS* ou *ULTRIX*) [5]. Conseqüentemente, aplicações paralelas podem ser desenvolvidas e executadas em redes locais heterogêneas. O processo de desenvolvimento e depuração de aplicações é bastante flexível utilizando–se os compiladores, depuradores e outras ferramentas próprias de cada máquina.

O *CPS* é composto por dois módulos, um para a programação e outro para a execução, depuração e monitoração de aplicações. O primeiro módulo é constituído por um conjunto de primitivas básicas escritas em linguagem C e oferece os mecanismos necessários para a implementação de processos cooperativos, tais como sincronização (barreiras), chamada remota a procedimento, transferência de mensagens, inicialização e término de processos. O segundo módulo, denominado *"Job Manager"*, é o processo responsável, entre outras tarefas, pela gerencia da aplicação incluindo inicialização e término de processos, alocação de recursos junto ao sistema operacional, redirecionamento das operações de E/S de processos em execução em máquinas remotas.

O projeto de uma aplicação *CPS* implica necessariamente três etapas: particionamento do algoritmo seqüencial em uma solução paralela composta de tarefas funcionalmente diferentes; codificação dos programas que farão parte da aplicação; e confecção de um arquivo chamado "Job Description File" que descreve uma configuração de execução da aplicação indicando número de processos a criar para cada tarefa e um mapeamento de alocação de processos para processadores. A parte mais complexa deste processo de projeto da aplicação paralela é o particionamento da solução em várias tarefas de execução paralela. Além da definição de cada tarefa, devem ser concebidas interfaces que determinem a interação entre tarefas de uma forma cooperativa.

4. Arquitetura da Aplicação

O emprego de paralelismo será efetuado nas três etapas mais importantes do processo de síntese das imagens:

- Etapa 1: no cálculo dos fatores de forma;
- Etapa 2: na resolução dos sistemas de equações lineares e
- Etapa 3: no "rendering" da imagem.

Estas etapas possuem características que proporcionam um elevado grau de paralelismo. O cálculo dos fatores de forma de cada "patch" do ambiente é independente do cálculo dos fatores de forma dos demais "patches". A resolução do sistema de equações para cada banda de interesse do sistema de cores também é efetuada de forma independente. Do mesmo modo, o algoritmo empregado na etapa de "rendering" calcula a intensidade de cada "pixel" de forma separada. A estratégia para a aplicação de paralelismo nesta última etapa consistirá basicamente na divisão da tela do dispositivo de exibição em regiões, sendo os cálculos correspondentes a cada região realizados em paralelo.

A aplicação proposta foi particionada nas seguintes tarefas:

- Tarefa 1: Processo "cliente" que inicializa estruturas de dados globais e coordena a execução de etapas de cálculo através de chamadas a subrotinas remotas implementadas nos processos definidos pelas tarefas 2,3 e 4.

- Tarefa 2: Implementa o serviço de cálculo de fatores de forma. Dependendo do número de estações de trabalho disponíveis podem ser criadas várias réplicas deste serviço.
- Tarefa 3: Implementa o método "LU Decomposition" de resolução de sistemas lineares de equações. Em um primeiro momento, são criados apenas três processos implementando a tarefa 3. Cada um fica responsável pela resolução de um sistema de equações correspondente a uma banda de interesse do sistema de cores. Esta abordagem, no entanto, limita as possibilidades de um ganho maior de desempenho, já que serão utilizados apenas três processadores da rede, permanecendo os demais subutilizados durante esta etapa do processo. A alternativa que será pesquisada e empregada em trabalhos futuros será a exploração de paralelismo interno ao método utilizado. Desta forma, um número maior de processadores será empregado e obter-se-á um melhor desempenho.
- Tarefa 4: Calcula a intensidade de cada "pixel" de uma determinada região do dispositivo de exibição de acordo com o mapeamento das coordenadas espaciais correspondentes à cena para as coordenadas de dispositivo. A tarefa 4 deve ser executada por vários processos.
- Tarefa 5: Realiza a exibição da imagem. Esta tarefa é executada por apenas um processo que recebe as intensidades dos "pixels" a serem exibidos diretamente dos processos servidores da tarefa 4.
- Tarefa 6: Dedicada à gravação do arquivo de fatores de forma e do arquivo imagem.

5. Resultados Obtidos

Na fase atual da implementação já foram obtidos resultados relativos ao cálculo dos fatores de forma. Os testes, cujos resultados são apresentados neste trabalho, foram realizados em uma base de "hardware" formada por uma rede composta por cinco estações de trabalho SUN Sparc 2. Os resultados obtidos correspondem à utilização exclusiva da rede. Em outras palavras, os processadores estavam alocados à execução dos serviços descritos neste trabalho e não estavam sendo utilizados simultaneamente por outros usuários.

O gráfico mostrado na Figura 1 apresenta os ganhos de desempenho em relação à aceleração, *"speed-up"*, correspondentes à execução seqüencial e às execuções paralelas do algoritmo de cálculo dos fatores de forma. Estes valores são confrontados com o "speed-up" ideal. Os ganhos de desempenho obtidos são praticamente lineares em relação ao número de processadores utilizados. Tomando-se como exemplo o emprego de três processadores, o "speed-up" obtido foi de 2,84.

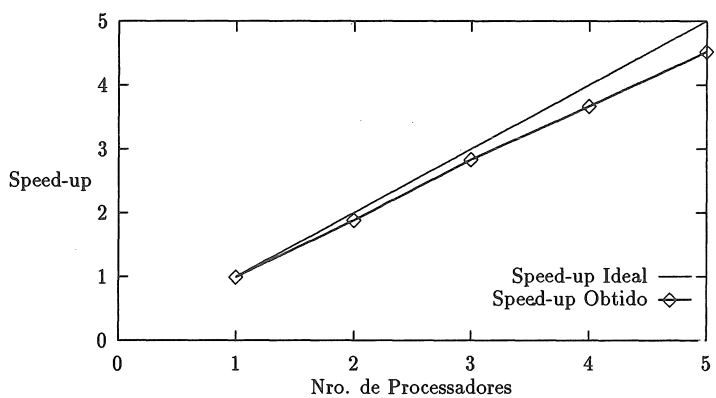


Figura 1. Speed-up x número de processadores.

Além da análise referente ao "speed-up", foi efetuada a comparação dos tempos correspondentes às execuções seqüencial e paralela das etapas do processo de síntese. Esta comparação baseou-se no tempo dispendido na síntese de uma imagem correspondente a um ambiente dividido em 992 "patches". Esta imagem, apresentada na figura 2, foi exibida em uma estação de trabalho SUN 4/330 "true-color" com uma resolução de 1152x900 "pixels".

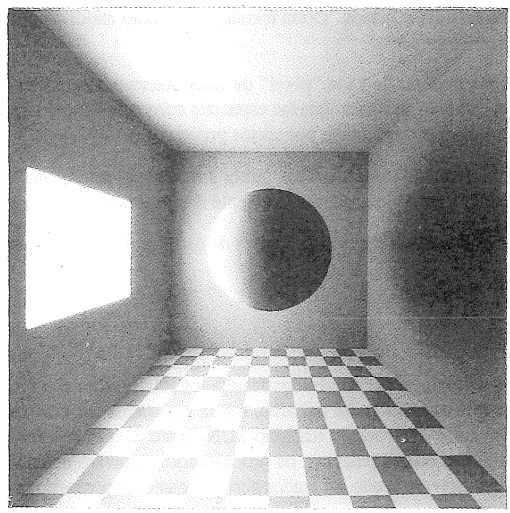


Figura 2. Imagem gerada empregando-se o método da radiosidade.

Os tempos de execução seqüencial de cada uma das três etapas descritas anteriormente são apresentados na Tabela 1. Estes valores ressaltam a importância do cálculo dos fatores de forma no processo de síntese das imagens, sendo responsável por 81,29% do tempo total. Por este motivo, neste trabalho, foi priorizada a paralelização desta etapa.

Tabela 1. Tempos de execução seqüencial das etapas do processo de síntese.

Etapa	Tempo de Execução	Fração do tempo total (%)
1	176m 36s	81,29
2	35m 56s	16,54
3	4m 43s	2,17

Os tempos de execução paralela das etapas do processo de síntese são apresentados na Tabela 2. Embora os valores correpondentes às etapas 2 e 3 sejam valores estimados, baseados nos valores de "speed-up" referentes à etapa 1, é indiscutível o ganho de desempenho que pode ser obtido aplicando-se a estratégia de paralelismo proposta. Na versão final da implementação da aplicação proposta visando a síntese de imagens em um ambiente de produção, as etapas 2 e 3 também serão implementadas por tarefas paralelas. Embora estas etapas não apresentem frações de tempo tão significativas quanto a etapa 1, a aplicação de paralelismo nestas aumentará a eficiência global do processo. Ainda, como imagens diferentes de uma mesma cena não necessitam de um recálculo dos

fatores de forma, a paralelização das etapas 2 e 3 torna-se, além de conveniente, necessária para que se possa obter imagens diferentes mais rapidamente.

Tabela 2. Tempos de execução paralela das etapas do processo de síntese.

Etapa	Tempo de execução	Nro. de Processadores	Speed-up
1	39m 1s	5	4,52
2	12m 1s	3	2,84*
3	1m 3s	5	4,52*

* valores estimados

6. Conclusões

A estratégia proposta, por ser baseada em "software", apresenta certas vantagens relevantes. Primeiramente não implica a utilização de uma base de "hardware" específica, permitindo o emprego de uma gama variada de equipamentos sem problemas de portabilidade. Além disso, permite que a aplicação proposta seja executada em máquinas de altíssimo desempenho, como o *ACP II*.

Na aplicação apresentada foi utilizada uma abordagem "full-matrix" do método da radiosidade. Esta abordagem tem como característica a execução seqüencial das etapas do processo de síntese. Por este motivo a estratégia proposta restringiu-se ao emprego de paralelismo dentro de cada etapa. A utilização de uma abordagem do tipo refinamento progressivo [12] poderá permitir também a exploração de paralelismo temporal. Neste caso obter-se-á um maior grau de desempenho pela conjugação harmoniosa de dois tipos de paralelismo:

- paralelismo real obtido em cada estágio através da replicação dos serviços e
- paralelismo temporal obtido com a sobreposição dos tempos de execução dos serviços executados em estágios diferentes.

Conforme mencionado anteriormente, os resultados apresentados neste trabalho correspondem à utilização de uma rede de nodos dedicados ao processamento da aplicação proposta. Resultados bem mais modestos são obtidos quando os recursos da rede são compartilhados por outros usuários. Além disto, a performance de aplicações interativas executadas em um nodo utilizado por um processo de aplicação paralela apresenta uma significativa degradação no desempenho, uma vez que os dois tipos de processos consomem muitos ciclos do processador. Como o *CPS* foi originalmente projetado para executar em um ambiente dedicado (multiprocessador ACP II), esse problema não é solucionado na atual versão da ferramenta. Portanto, a execução da aplicação proposta em uma rede local de processadores não dedicados exigirá a utilização de um mecanismo de balanceamento dinâmico e adaptativo de carga, como o proposto por Garcia [11]. Este mecanismo considera aspectos tais como capacidade de processamento de cada processador, tempo de execução dos serviços e gerenciamento de carga imposta por outros usuários da rede.

Os ganhos de desempenho já obtidos e esperados justificam o emprego da estratégia proposta. Entretanto, para a utilização de redes com grande número de nodos, torna-se necessário o projeto cuidadoso de tarefas e serviços a fim de que se possa obter o melhor desempenho possível. Em trabalhos futuros serão apresentadas análises relativas ao comportamento da aplicação em redes heterogêneas de alto potencial de processamento formadas por um grande número de processadores.

Referências

1. C.M. Goral, K.E. Torrance, D.P. Greenberg and B. Battaile. "Modeling the Interaction of Light Between Diffuse Surfaces". Computer Graphics, Computer Graphics, 18(2) pp. 213-222 (1984).
2. M.Feda and W.Purgathofer. "Progressive Refinement Radiosity on a Transputer Network", In: Proceedings of the Second EUROGRAPHICS Workshop on Rendering, (1991).
3. A.G.Chalmers and D.J.Paddon. "Parallel Processing of Progressive Refinement Radiosity Methods", In: Proceedings of the Second EUROGRAPHICS Workshop on Rendering", (1991).
4. T.Nash, H.Areti, R.Atac, J.Biel, A.Cook, J.Deppe, M.Edel, M.Fischler, I.Gaines, R.Hance, D.Husby, M.Isely, E.Miranda, T.Pham, T.Zmuda, E.Eitchen, G.Hockney, H.B.Thacker, P.Mackenzie and D. Toussaint. "Software for Event Oriented Processing on Multiprocessr Systems", In: Proceedings of the Symposium on Recent Developments, Processors and Software for High-Energy Physics. Guajanato, Mexico (1984).
5. T.Nash, H.Areti, R.Atac, J.Biel, A.Cook, J.Deppe, M.Edel, M.Fischler, I.Gaines, R.Hance, D.Husby, M.Isely, E.Miranda, T.Pham, T.Zmuda, E.Eitchen, G.Hockney, H.B.Thacker, P.Mackenzie and D. Toussaint. " High Performance Parallel Computers for Science: New Developments at the Fermilab Advanced Computer Program", In: Proceedings of the Workshop on Computational Atomic and Nuclear Physics at One Gigaflop, (1988).
6. Fermi National Accelerator Laboratory. "ACP Cooperative Processes Software - User's Manual", Universities Research Association, (1989).
7. T.Nishita and E.Nakamae. "Continuous Tone Representation of Three Dimensional Objects Taking into Acount of Shadows and Interreflections". Computer Graphics, Computer Graphics, 19(3) pp. 23-30 (1985).
8. G.V.G.Baranoski. "The Parametric Differential Method: An Alternative to the Calculation of Form Factors", In: Proceedings of EUROGRAPHICS Conference and Exhibition, (1992).
9. W.H. Press, B.P.Flannery, S.A.Teukolsky and W.T.Vetterling. *Numerical Recipes in C The Art of Scientific Computing*, Cambridge University Press, 1988.
10. J.D.Foley, A. Van Dan, S.K. Feiner and J.F. Hughes. *Fundamentals of Interactive Computer Graphics*, Reading, Addison-Wesley, 1990.
11. L.C.Garcia. "Balanceamento de Carga em Sistemas Paralelos Fracamente Acoplados", CPGCC - UFRGS, Porto Alegre, 1992. (Dissertação de mestrado em andamento)
12. M.Cohen, S.E.Shenchang, J.R.Wallace and D.P.Greenberg. "A Progressive Refinement Approach to Fast Radiosity Image Generation". Computer Graphics, Computer Graphics, 22(4) pp. 75-84 (1988).

Agradecimentos

Os autores deste trabalho gostariam de agradecer aos pesquisadores do *Laboratório de Física Experimental de Altas Energias - Centro Brasileiro de Pesquisas Físicas (LAFEX/CBPF)* e do *Fermi National Accelerator Laboratory (FermiLab)* por terem viabilizado a utilização do *CPS* e aos colegas Valdir Belmonte, Roberto Amboni e Marinho Pilla Barcelos do CPGCC-UFRGS pela sua colaboração na revisão deste trabalho.

SELECTING VECTORS FROM A TRAINING SET FOR BACKPROPAGATION NETWORKS

Angel Gutierrez
Department Mathematics and Computer Science
Montclair State College
Upper Montclair, NJ 07043, USA

ABSTRACT

Backpropagation neural networks could be used to learn a mapping f from R^n to R^m. A set of examples, the training set, is available to achieve this goal. The learning process is carried out by presenting, to the network, elements from the training set, and modifying the weights according to the error on the output. These elements are selected using a fixed probability density function. This paper presents a new method, assuming that there is a uniform probability density function, but taking into account the varying difficulty of learning for each component. This new method produces smaller errors on almost all the cases.

Keywords : Neural networks, backpropagation networks, vector-valued functions, selection on training sets.

INTRODUCTION

A neural network with a regular configuration, according to [7], of two input nodes, two output nodes and a hidden layer containing s nodes, - see Figure 1 - is used to learn the mapping $f: A \subsetneq R^2 \dashrightarrow R^2$, $y = f(x)$, that represents the conversion from Cartesian to polar coordinates. If the neural network output is represented by $Y(x,w)$, where w is the set of weights associated with the nodes of the network, then a measure of the error between the actual values $f(x)$ and the one supplied by the network is given, according to [3], by

$$F(w) = \lim_{N\to\infty}\left(\frac{1}{N}\right)\left(\sum_{k=1}^{k=N} \|f(x_k) - Y(x_k, w)\|^2\right) \qquad (1)$$

The learning process, see reference [4], is accomplished by changing the value of the weights during the training period, according to the formula

$$w_{ji}(k+1) = w_{ji}(k) - \eta_k \times \frac{\partial F}{\partial w_{ji}} \qquad (2)$$

that is the known jump every time variant of the generalized delta learning rule [5],[10]. The generic weight w_{ji} of the network represents the weight associated with the information leaving the node i and entering the node j in the forward mode, and η_k is the learning rate at the step k. The output of the hidden layer is filtered through the sigmoidal function

$$y = \frac{1.0}{1.0 + e^{-x}}$$

to control the nonlinear terms, and the output layer provides the approximation to the two-dimensional mapping under consideration. We add a bias term to both input and hidden layers to make the neural network more powerful.

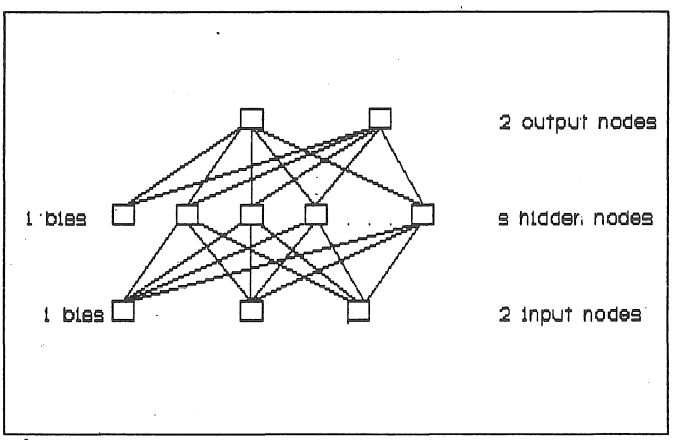


Figure 1 : Neural network

The vector-valued function is seen by a network as a single entity, and the vectors in the training set have the same probability to be chosen. It is reasonable to think that if we choose the vectors taking into account the varying difficulty of learning for each component, a better

result could be obtained. Therefore, when choosing vectors form the training set, we use a technique that resembles an inverse simulated annealing [1]. Each training vector is assigned a probability of acceptance based on its difficulty to be learned. Then each vector is rejected with a low uniform probability, unless its acceptance function removes the rejection.

This idea is studied in this paper varying the number s of nodes in the hidden layer in one case and changing the seed for the random number in another. In the next two sections we set up the working conditions for our examples and exhibit the obtained results. In the last section the conclusions are drawn.

SENSITIVITY ANALYSIS

The square $A \equiv \{ (x,y) \mid -1 \leq x \leq 1 , -1 \leq y \leq 1 \}$ is used as the original domain for the conversion mapping to be learned. So given $(x,y) \in A$, the map under consideration is

$$radius=\sqrt{x^2+y^2}$$

and

$$\theta = \begin{cases} \arctan\frac{y}{x} & for\ x>0, y>0 \\ \arctan\frac{y}{x}+2\pi & for\ x>0, y<0 \\ \arctan\frac{y}{x}+\pi & for\ x<0 \end{cases}$$

with obvious values for the 4 semiaxes.

Each network, containing s regular nodes and one bias node in the hidden layer is trained with a fixed set of T points (vectors) in the unit square. The networks were first trained using the correct output for each input vector, using a uniform probability density function. Then, in order to replicate the experimental condition the same networks where trained with the new method. When each learning process was completed, i.e. the number of predetermined presentations were concluded, the error in learning the training set was computed, using the

formula 3,

$$e = \frac{1}{T}\left(\sum_{k=1}^{k=T} \| f(x_k) - Y(x_k, w) \|^2\right) \qquad (3)$$

which is a modification of the mean square error function, where x_k are the training points. Since overtraining is a normal phenomenon in backpropagation neural networks, the trained networks were tested with a set of P new points. For this new set we compute the new error, using a formula equivalent to (3), with T replaced by P, i.e.

$$e_0 = \frac{1}{P}\left(\sum_{k=1}^{k=P} \| f(x_k) - Y(x_k, w) \|^2\right) \qquad (4)$$

At the same time, since the new method is supposed to have different learning ability for each output component, two new errors are computed, according to the formula

$$e_j = \frac{1}{P}\left(\sum_{k=1}^{k=P} | f_j(x_k) - Y_j(x_k, w) |^2\right) \qquad j=1,2 \qquad (5)$$

where e_1 represents the error on the radius, and e_2 is the error on the angle.

Each training vector was selected if the random uniform variate obtained for it was below q, a fixed value during the training, with $0 < q \leq 1$. If the vector was rejected by the previous criterion, it could be accepted if a new variate was below f(p), with $0 < f(p) \leq 1$. Here f(p) is a given function, which depends upon p, a variable that in some way denotes the learning difficulty associated to that vector.

EMPIRICAL RESULTS

The experiments were conducted using a set of 500 training points. The learning rate η was given by the linear function

$$\eta_k = \eta_0 + (\eta_1 - \eta_0) \times \frac{k}{100000}$$

where k is the iteration step, η_0 is the initial learning rate, set at 0.050, and η_1 is the final rate, set at 0.001. These values for the initial and final learning rate were known to be

acceptable [2]. The guided selection of the training set was established by the following function

$$r<q \vee s<f(p)$$

where r and s are random uniform variates and

$$f(p)=1+p\times(-1.2+0.6\times p)$$

p being defined as

$$p=\frac{a}{\pi}$$

and a is the angle associated to the training vector. In other words, we are assuming that the angle is more difficult to be learned when it is near 0 and 2π radians. The function f(p) tries to represent quantitatively this difference, but it was chosen arbitrarily. If the new method worked with this unsophisticated function, it gives a guarantee that it will work if a more elaborated function is used.

The testing set contained 18018 points chosen in the following way : The radius was changed from 0.1 to 1 using increments of 0.01; for each value of the radius the angle will take two set of values. The first set contains the angles from $\pi/51$ to $99\pi/51$ using increments of $\pi/51$ radians. The second one contains the angles from $\pi/50$ to $99\pi/50$ using increments of $\pi/50$ radians. Using a large set of testing vectors will avoid the appearance of sampling errors in our computation of the quadratic and component errors.

Since a network with more hidden nodes does not necessarily perform better [6],[10], the experiment was started using 24 hidden nodes, and then the number of nodes was diminished, until a configuration that contained 12 hidden nodes was reached, see reference [2].

In this first experiment the hidden nodes were changed, but the seed for the random number generator was 123451, the same for all of them. The number of presentations of the training vectors also remained constant with a value of 100000. The weights were initialized using

random values in the interval [0,1). The results of this experiment are shown on Table 1, although the error e_0 in this and all the other tables is shown with a magnification of two, to see the differences more clearly. That means that the actual error will be obtained from the table by dividing by two.

s	e	e_0	e_1	e_2	q	e	e_0	e_1	e_2
24	0.156	0.265	0.046	0.276	0.8	0.153	0.257	0.046	0.276
20	0.153	0.259	0.036	0.284	0.8	0.154	0.259	0.035	0.281
18	0.141	0.247	0.034	0.264	0.8	0.141	0.231	0.042	0.261
16	0.176	0.310	0.066	0.286	0.8	0.179	0.310	0.068	0.286
14	0.189	0.351	0.084	0.303	0.8	0.190	0.348	0.083	0.303
12	0.215	0.313	0.075	0.272	0.8	0.167	0.282	0.064	0.275

Table 1

In the second set of experiments, the number of hidden nodes was 14, but now the random seed was changed. The number 14 was chosen because it represents, according to [8], a good value for our number of training points. The number of presentations of training vectors was increased to 200000 and the weights were initialized using random values in the intervals [0,1) and (-1,0]. Finally the value of q was changed to 0.9 in all the guided learning cases. In this case the results can be found on Table 2, 3 and 4, where the values on the left are obtained by the regular training method, and the values on the right are obtained with the new method.

e	e_0	e_1	e_2	Seed	e	e_0	e_1	e_2
0.160	0.250	0.031	0.274	772241	0.156	0.241	0.036	0.266
0.128	0.202	0.044	0.216	335511	0.129	0.199	0.042	0.216
0.148	0.252	0.064	0.234	554431	0.147	0.241	0.059	0.233
0.112	0.182	0.033	0.208	643231	0.113	0.181	0.033	0.203
0.124	0.206	0.036	0.231	16777216	0.123	0.202	0.036	0.231
0.146	0.229	0.053	0.233	223341	0.145	0.225	0.052	0.232

Table 2

e	e_0	e_1	e_2	Seed	e	e_0	e_1	e_2
0.153	0.250	0.051	0.250	3445423	0.157	0.250	0.051	0.252
0.130	0.213	0.053	0.219	225543	0.130	0.206	0.052	0.212
0.118	0.186	0.035	0.216	224453	0.118	0.183	0.031	0.215
0.127	0.215	0.026	0.246	2244131	0.128	0.211	0.025	0.238
0.130	0.222	0.055	0.228	225533	0.131	0.217	0.055	0.228
0.117	0.200	0.032	0.222	2125533	0.115	0.197	0.033	0.222

Table 3

e	e_0	e_1	e_2	Seed	e	e_0	e_1	e_2
0.135	0.226	0.065	0.213	131	0.135	0.224	0.066	0.209
0.150	0.245	0.063	0.233	13457	0.149	0.241	0.063	0.231
0.150	0.223	0.028	0.247	11334457	0.152	0.232	0.028	0.255
0.125	0.205	0.036	0.223	331155	0.120	0.191	0.032	0.215
0.165	0.265	0.053	0.255	334551	0.162	0.257	0.049	0.263
0.102	0.175	0.027	0.207	345523	0.102	0.175	0.026	0.206

Table 4

CONCLUSIONS

A new method to select vectors from a training set has been compared with the regular one. The results on all tables corroborate the conjecture that the new method performs equal to or better than the regular one. The only exception occurs in table 4, for the seed 11334457. The value of these errors lies in the middle range of all found errors, 0.175:0.265. When the value of q was diminished to 0.8 for this particular case, the error of the new method diminished to 0.230, but still remained greater than the one obtained using the regular method.

In the cases where the errors e_0 were the same, like in table 4, case of seed 345523, using a new value of q = 0.8, the error e_0 was diminished to 0.170, with e_1 = 0.024 and e_2= 0.204, an improvement over the previous results. Therefore it can be said that the lowest errors of all the configurations correspond to a network trained with the new method using a q value of 0.8.

The results of the experiments are promising, but more detailed comparisons are to be made to determine the influence of the function f(p) chosen, of the value of q, and of the number of presentations, on the behavior of the networks with respect to each method.

BIBLIOGRAPHY

1. I.O. Bohachevsky, M.E. Johnson and M.L. Stein,"Generalized Simulated Annealing for Function Optimization", Technometrics, 28, 209-217, 1986.

2. A. Gutierrez, "Learning Simulation of Vector-Valued Functions", Proc. of the Twenty-Second Annual Pittsburgh Conference on Modeling an Simulation, 22, 1938-1943, 1991.

3. R. Hecht-Nielsen, "Neurocomputing", Addison-Wesley, 1989.

4. R. Hecht-Nielsen, "Theory of the Backpropagation Neural Networks", Proc. of the International Joint Conference on Neural Networks, I, 593-611, 1989.

5. M. Hirsch, "Dynamical Systems Review", tutorial presented at the 1988 IEEE International Conference on Neural Networks.

6. E.A. Rietman, "Experiments in Artificial Neural Networks", Tab Books Inc., Blue Ridge Summit, PA, 1988.

7. D.E. Rumelhart, G.E. Hinton and R.J. Williams,"Learning Internal Representations by Error Propagation", in Parallel Distributed Processing: Explorations in The Microstructure of Cognition, Cambridge, MA, MIT Press, 1986.

8. SARA: Entorno para el Desarrollo de Aplicaciones de Redes Neuronales, Manual del Usuario, ADIC-IIC, Madrid, 1992.

9. F.M. Silva and L.B. Almeida,"Speeding up backpropagation", in Neural Networks for Sensory and Motor Systems, R. Eckmiller (ed.), North-Holland, 1990.

10. H. White, "Learning in Artificial Neural Networks: A Statistical Perspective", Neural Computation, 1, 425-464, 1989.

A Cost Evaluator for Parallel Deductive Database Systems

A. Hameurlain, E. Ceccato, F. Morvan,

Université Paul Sabatier, Lab. IRIT
118 Route de Narbonne -31062 Toulouse - France
Tél :+33 61 55 67 65 ; Fax : +33 61 55 62 58 ; E-mail : hameur@irit.fr

Extended Abstract

1. Introduction and motivations

New applications such as office information system, geographical information systems, CAD, expert systems and other applications which require large knowledge bases, need improved efficiency in response time from advanced database management systems (DBMS). This goal can be achieved by optimizing SQL query (eventually recursive [Ban86]) compilers and using optimal parallel execution strategies for query processing [Val88].

The conception of an optimizer is fundamental to obtain high performance. The rôle of the optimizer for recursive or non recursive SQL queries for parallel execution is to generate an efficient execution plan to get the information requested by the user. Thus, the optimizer must be capable of selecting one of several execution plans with highest performance. To estimate query processing costs, the optimizer needs another system called cost evaluator. However, the specificities of the hardware and software environment are usually not taken into account by the optimizer with classical cost models. Indeed, each DBMS contains a specific module for estimating query costs, which is not compatible with other systems. An elegant solution consists in separating the optimizer and the cost evaluator [And91]. The advantage of this solution is to provide an *extensible, flexible and adaptable* cost evaluator. The cost evaluator is seen as an intelligent black box by the optimizer and can therefore be used independently by other DBMS. This design leads to the parametrization of the cost evaluator with libraries describing the environment (Architecture, database profile, cost models). The cost evaluator proposed previously by [And91] only applies to the optimization of SQL queries for parallel execution. We therefore extended this evaluator to take into account the optimization of recursive queries as follows:

(i) We define and integrate a new parameter which evaluates the number of new tuples deduced by the specialized operators (i.e. Transitive Closure [Ioa88], External Clo-

sure,...). With knowledge of the number of tuples, the number of processors can be estimated [Ham92] for the optimal parallel execution of a relational or specialized operator.
(ii) We extend the database profile by adding the *canonical data structures* associated with each referenced base or deduced relation into premises of the recursive rules of the intentional database.
(iii) We integrate a new parameter, called *acceleration factor* Ω, to evaluate the benefit from the propagation of the attributes and the number of processors during the parallelization process.
(iv) We include in the cost model a *decision table* to choose the best of sequential or parallel algorithms for each execution strategy.

In this paper we briefly recall the various steps in the optimization of recursive or non recursive SQL queries. This points out the contribution of the cost evaluator to the optimizer. Then we describe the functions of a cost evaluator for parallel deductive systems. Finally, we explain the advantage of a multi-environment oriented cost evaluator working independently from the optimizer.

2. Query Compiler Architecture for a Parallel Deductive System

The compilation process consists of several stages up to the generation of an optimal program, which may be parallel, to be executed on the algebraic machine (Cf. Appendix). Construction of such a program proceeds as follows:

(a) Analyser and Resolution Process: The SQL query is analysed syntactically and semantically then transformed in an execution plan in the shape of resolution graph [Ham90].

(b) Optimization: The resolution graph is restructured, during the logical optimization, by applying the transformation rules, as well as the algorithm for selection propagation [Agr89][Ham90][Kif86] in the recursive queries sub-graphs. The joins are ordered, during the physical optimization by using the keys and indexes stored in the *database profile*. Thereafter the resolution graph is enriched with new information such as the *number of processors*, the *local response time* calculated by the cost evaluator during the parallelization phase of relational and specialized operations (Transitive Closure, External Closure). The last phase consists in scheduling in an eventually parallel fashion all the operations of the execution plan in order to minimize the query response time.

(c) Code generation: The optimal resolution graph enriched with the control and communication operations is translated by the code generator into a code "object" which is executed by the algebraic machine on the various nodes of the parallel architecture.

3. Functional aspects of the cost evaluator

Let us first define the various *metrics*, i.e. costs, to be calculated by the evaluator. Some representation of these costs is required and the evaluator must know the value of each parameter in the algebraic machine environment. These data are stored in *libraries* we will describe later.

The different metrics are:

(i) The number of processors NP: For each resolution graph operation, the evaluator must determine the best number of processors to minimize response time. But response time is mainly function of the size of the operand relations. First the evaluator must calculate the number of deduced tuples. Other parameters such as tuple production, distribution or transfer times are stored in libraries.

(ii) The local response time LRT: As a function of the algorithm chosen for the operation, the evaluator may determine the local response time thanks to a set of cost functions (communication, distribution and effective time). Amongst the parameters implicated in these functions, some are stored in the libraries (architecture, BD profile), others have been determined previously (numbers of processors and of deduced tuples).

(iii) The acceleration factor Ω*:* During the parallelization phase, the optimizer must determine whether propagation is beneficial or not. The acceleration factor is calculated by the evaluator and defined as the ratio between local response times with and without propagation.

The values of these parameters are stored in the following libraries:

(1) Architecture: This data structure contains the information related to all types of architecture (memory, cpu, network) even if only a given architecture is being used at a given moment. This ensures the evaluator may be used in any environment.

(2) EDB&IDB Profile: This contains information on the extentional database EDB (relations, attributes, keys, indexes,...) and intentional IDB one concerning the canonical data structure of the base or deduced relations used in the definition of the recursive relations. This canonical data structure is represented by a set of parameters which are: Fin, Fout, expansion factor.

(3) Cost models: This contains, on one hand, the set of cost functions which allow the evaluation of the metrics defined above and on the other hand, a decision table which allows selection of the best algorithm for each operation as a function of some data (size of the relations...)

4. Conclusion

In this paper we presented a multi-environment cost evaluator, independent from the optimizer. The main advantages of separating the optimizer and the cost evaluator are:

(i) Extensibility: new environments (architecture, DB_profil, access methods, new algorithms) can be taken into account simply by extending the libraries.

(ii) Flexibility: the evaluator has a structure well dissociated from the optimizer. The volume of information it contains can therefore be increased without any modification of the optimizer. Furthermore, the internal structure of the cost evaluator is also flexible since new parameters can be integrated and defined all along without modifying its structure.

(iii) Adaptability to various database Machines. To use the cost evaluator it suffices to call it with its environment. Thus, the environment can be changed without having to rewrite the software.

Appendix

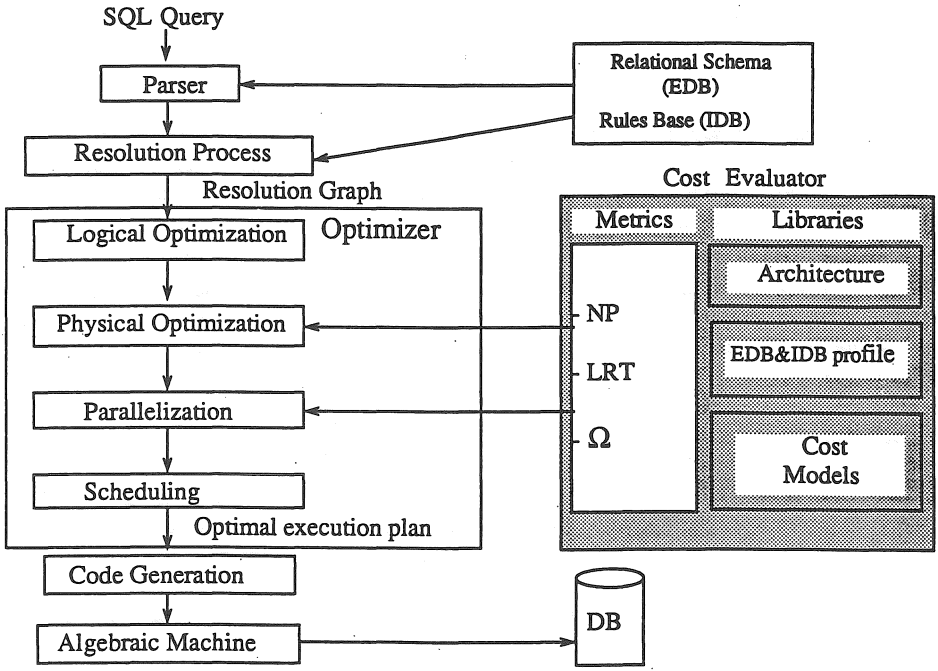


Figure 1 : Architecture of the SQL query compiler

References

[Agr89] R. Agrawal, P. Denabu, "Moving Selections into Linear Least Fixpoint Queries", IEEE Transaction Knowledge and Data Engineering. Vol. 1, No. 4, Dec 89, pp. 424-432.

[And91] F. Andrès et al., "A Multi-Environment Cost Evaluator for Parallel Database Systems", 2nd Intl. Symp. on Database Systems for Advanced Applications, DASFAA'91, Tokyo, April 1991.

[Ban86] F. Bancilhon, R. Ramakrishan, "An Amateur's Introduction to Recursive Query Processing Strategies", Proc. of ACM- SIGMOD, Washington, May 1986, pp. 16-52.

[Ham90] A. Hameurlain, F. Morvan, "An Algorithm For Selection Operator Propagation in Resolution Graph", Intl. Conf. On Database and Expert Systems Applications, Vienna, Aug. 1990, pp. 550-553

[Ham92] A. Hameurlain et al., "Processor Number Estimation for Optimal Parallel Execution of recursive Queries", European Workshops on Parallel Computing, Barcelona, March 1992.

Ioa88] Y.E. Ioannidis, R. Ramakrishan, "Efficient Transitive Closure Algorithm", Proc. of the 14th VLDB Conf. Los Angeles, California, pp. 382-394, 1988.

[Kif86] M. Kifer, E. Lozinskii, "Filtering Data Flow in Deductive Database", Proc. of the Intl. Conf. on Database Theory, Lecture Notes in computer Science, No. 243, Springer-Verlag, Rome, Sept 1986.

[Val88] P. Valduriez, S. Khoshafian, "Parallel Evaluation of the Transitive Closure of a Database Relation", Intl. Journal of Parallel Programming, Vol. 17, No. 1, Feb. 1988.

USING COMMUNICATIONS AND MULTIMEDIA TECHNOLOGIES IN A NEW APPROACH FOR COMPUTERS IN EDUCATION IN DEVELOPING COUNTRIES

Pedro Hepp K., Ph.D.
María Inés Alvarez H.
Enrique Hinostroza S.
Ernesto Laval M.

Pontificia Universidad Católica de Chile - Escuela de Ingeniería
Ministerio de Educación Chile - Programa MECE

Casilla 306 (143) Correo 22 Santiago, Chile
Phone: (56)2 5522375 x4440
FAX: (56) 2 5524054
e-mail: INTERNET ique@lascar.puc.cl

ABSTRACT

The use of computers in elementary education in developing countries has been usually centered on learning about computers and on using programming languages and a few unproven educational software packages. Modern trends emphasize the computer as a media or as an empowering tool for teachers and students, centering the focus on the people that use it and on ways of using the technology as a means of making the school a more attractive and effective place. This paper presents a national project that is presently under way in Chile and which was designed on three basis: the knowledge about the people and schools that will eventually use the technology, the reported world wide experience in computers and education of the last decade and the new opportunities of modern technologies, mainly multimedia, communications and human-interfaces. The project started as a prototype in 1991 and will gradually expand during the next 5 years.

Keywords: Computers in Education, Educational Network, Communication, Multimedia, Human Interface.

INTRODUCTION

A major effort in education is under way in Chile. The World Bank and the Chilean Government have designed a 5 year Program whose key objective is to assist the Government in enhancing the efficiency, quality and equity of primary education in selected schools in targeted urban and rural low-income areas. Part of this large Program is a project that considers the use of computers and communications in some of those schools.

During the project's design, three major aspects were considered:

1. The chilean educational reality: its high degree of administrative centralization, a high number of rural and mainly isolated schools, poorly motivated teachers, school abandonment, low educational performance indexes, few opportunities for local innovation, etc. The cultural, geographical, social and ethnical diversity was also considered [2].

2. The world wide experience in computers in education [8] and its status in Chile was revised in order to assess large scale efforts and to look for trends in usage, training, technology and evaluation mechanisms. The last decade was regarded as one with high expectations and few significant results in terms of the role, value or effectiveness of computers in schools [1]. In Chile almost 30% of schools in Santiago have 8 bit computers and very few software. Teachers are mainly self-trained [3] [4].

3. The new opportunities of modern technology. Computers of the nineties include enhanced capabilities allowing new ways for using them in schools. In particular multimedia [5] [6] -integrating text, sound, animation and live video in a multidimensional associative space- and communications [7], allowing the work in collaborative projects among geographically dispersed schools children and teachers [8]. In this respect, many southamerican countries are rapidly improving and expanding their telephone systems, and some, like Chile, are installing large fiber optics networks.

These three considerations led us to propose the following objectives for the project:

1. Gradually develop a computer network comprising elementary schools and research centers in four chilean regions, using the public telephone system or radio. It is also expected to communicate with foreign schools using the Internet facilities as a gateway. The project has started connecting 5 nodes in Santiago and 1 in U.S.A. in 1991 and will gradually expand until it reaches about 100 schools covering a variety of cultural, geographical, social and ethnical communities. It is also expected that at least 10 research centers (universities and institutions) related to education and community development initiatives will join the network to participate in collaborative projects.

It is important to emphasize the effort on providing very simple to use human-interfaces, building an abstraction over the operating system and the communication platform. Training on how to become a computer user and how to take advantage of the communication system and the educational software will have a low profile. The experience conducted so far demonstrated that carefully chosen graphical metaphors can dramatically reduce computer training time. Training will be devoted to methodological rather than to technical issues.

2. Install an "Educational Technology" Laboratory in a regional university to support, design and stimulate the projects and activities that will be carried on through the network. This laboratory will have the resources, the people and the equipment to build and test multimedia educational software. It will be also the place where the evaluation of the experience will be carried on. The laboratory will be staffed by an interdisciplinary team, comprising software and communication engineers, teachers, psychologists, graphics designers and social workers.

THE PLAZA

The Project will officially start in early 1992 but a prototype network was implemented in Santiago comprising 6 nodes: 3 local elementary schools, the Catholic University of Chile, Apple Computer headquarters in Santiago and a school in Albuquerque, New México. The schools in Santiago were 2 public schools in low-income areas and the other one was a private high-income school.

Plazas are places well known in Spain and Latin-America. A Plaza is an attractive space for children, a place for getting together and to start interesting adventures. The following picture is a crude black and white version of the full-colored original interface.

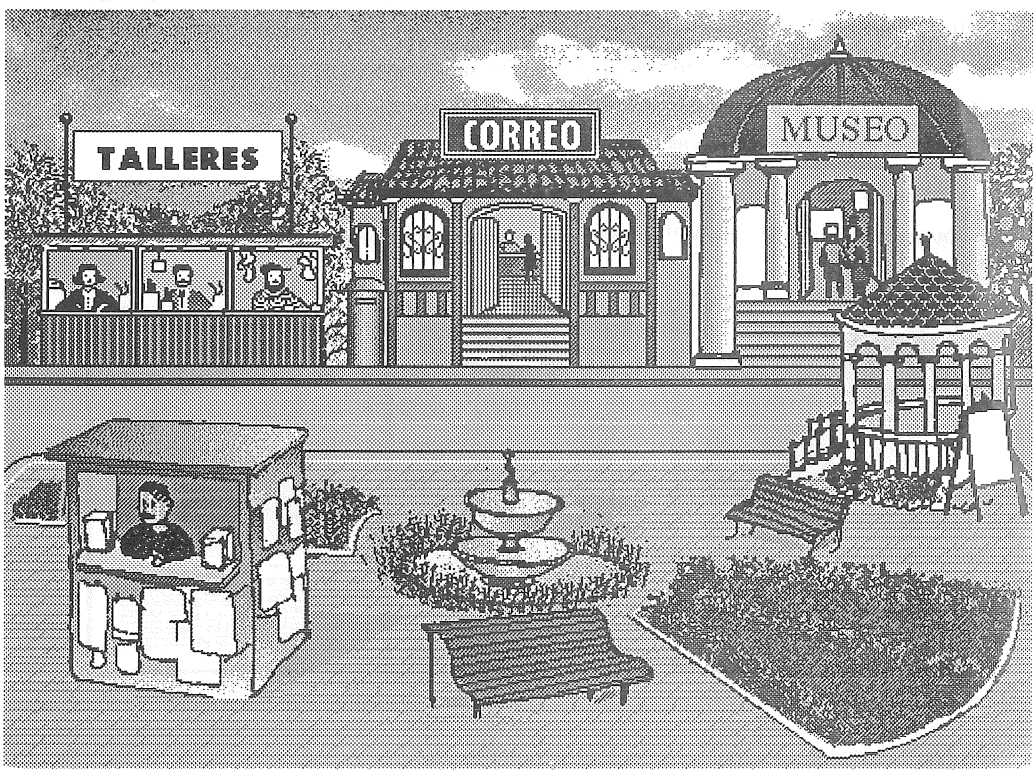


Figure 1. A b/w version of The Plaza

The Plaza is the human-computer interface that a teacher or student encounters in the computer. There are no command languages involved in using it, nor is it necessary any particular training, only pointing and clicking with the mouse. Its main purposes are to be inviting and to offer a simple window to the available functionality. Teachers and students were exposed to a short demonstration and were left on their own. In most schools, the computers were first left to the teachers for at least a week in order to allow them to get confident with the

system and prepare them for trouble shooting. Most problems and questions were solved using the e-mail system itself without having to go to the schools.

The Plaza has the following parts, each one being a well known metaphor with a particular meaning and purpose:

1. The Kiosk

The Kiosk offers a window to an information space comprising newsletters, stories and educational comics. Students can read their newspapers, which are collaboratively build by several children in different schools. They have a lively format and contain their own and also some centrally provided news. A central editor exists at the laboratory who receives the contributions from the users. Similarly, the teacher's newspaper has a central editor and contains their news and also general purpose articles. The newspapers offer presently only text and graphics and very crude animations but is expected in the future that they will contain better animations and short video sequences.

The Kiosk includes also short stories and comics as a stimulus for reading. Students are encouraged to send their own illustrated stories which, after some editing can be found in the Kiosk.

Figure 2. B/W view of newsletters and stories available in the Kiosk

2. The Museum

The museum is a friendly interface for a simplified distributed database of educational software applications and general purpose articles. Users can browse through an index and select a piece of software. If the selected material is on place, it is run for the user -generally a multimedia educational application-, if it is not, a message is sent to the central server in order to deliver the material the next day. The next figure shows a black and white portion of the Museum's interface:

Figure 3. B/W version of parts of the Museum's interface.

Presently, less than 10 applications are available at the museum but one of the project's goal is to gradually build a comprehensive library of tested software. Most of this software will be built in the laboratory in collaboration with teachers or directly by the teachers themselves. Only proven, robust and well designed applications will be allowed into the Museum's collections. Each application will have a curriculum, stating its ownership, users, experience, recommendations and whom to write to share or ask information about it. The section on general purpose articles presently contains only a few educational articles in electronic format, available to the teachers. It should also gradually evolve into an electronic library, to be used in the same way as the applications.

3. The Workshop

Teachers and students will be encouraged to work on collaborative projects (i.e. story writing, scientific experiments) and to build their own multimedia applications using general purpose authoring tools. The results of these efforts will be delivered to the museum. These activities are organized and conducted in the Workshop which contains an information space for every interest area, as in some bulletin boards. Users build their project and use the e-mail system from inside the Workshop.

4. The Post Office

The post office is presently a simple to use bulletin board. Although all letters are broadcasted to all schools, the Post is separated in two areas, one for teachers and one for students.

Users can presently write and draw pictures on their letters. The main purpose of the Post Office is to establish initial contacts with people having similar interests in order to later continue their communication under a Workshop project.

The following figure shows a letter sent to everyone ("Todos") by a group of children ("Los Bombachos") of the "San Joaquin" school the 10th of december 1991.

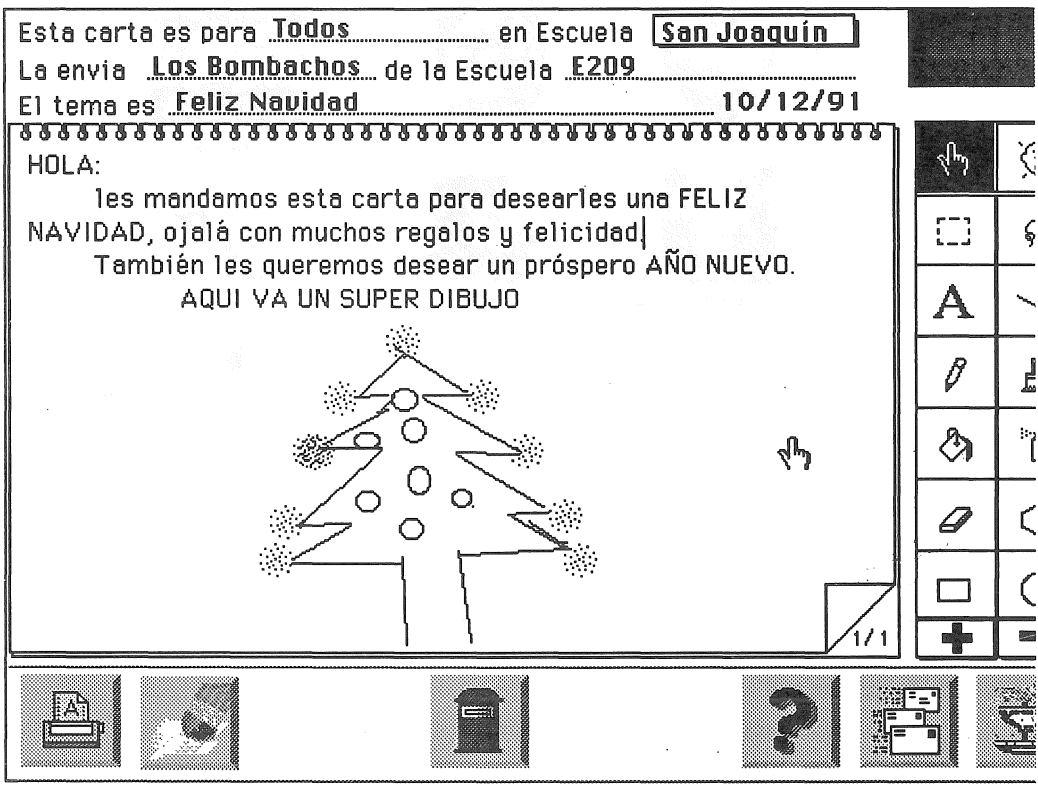


Figure 3. A b/w version of a letter sent.

ACTIVITIES ON THE PLAZA

Introducing the computers in one primary school that had never used computers was easier than expected and presented less problems than in the two schools with computers. After a 20 minutes demonstration to teachers and a 5th grade class on how to navigate through the Plaza, the users were left on their own. Most of the problem encountered were due to user-interface design flaws and provided an important feedback for the next step in the project.

Children were encouraged to establish pen-pal relations with children from other schools. This succeeded rapidly and it was proved that a well designed e-mail system encourages the reading and writing habits. After a few days of writing letters and drawing pictures, students

were participating in different activities. Some of them were playing chess across the network, working on a cultural quiz, participating in a drawing competence, etc.

Although the experience lasted for only a few months, all students and all teachers participating regarded the project as highly motivating and would enthusiastically continue in 1992. The designers goals were obtained in terms of validating the user-interface, demonstrating the usability of the public telephone system (at between 2400 and 9600 bps, most of the time) and the attractiveness of multimedia software for both teachers and students. All activities on the Plaza were recorded for future analysis.

FUTURE WORK

Future work will concentrate on several areas.

1) the user interface has to be revised, making a more lively Plaza. There are several opportunities to introduce meaningful multimedia capabilities, such as moving icons when e-mail arrives, sound or animations when newsletters are updated. The Plaza is regarded as a first step toward an "educational town". For example, a health care center will be provided to learn about personal hygiene, eating habits. Also, the underlying communications architecture will have to be improved, including more efficient ways of connecting the schools and the ability to handle larger school networks.

2) a teacher training strategy will have to be implemented using mainly short in-site workshops and the network for follow-ups. Although the software is very simple to learn, many teachers are frightened anyway, particularly if they have been exposed to older command driven interfaces. Most of the training will be devoted to methodological issues, such as how to prepare, conduct and evaluate an experience with students using computers.

3) the place for the computer at the schools must be carefully chosen. It seems that the library and the teacher rooms are strong candidates for extracurricular uses.

4) One of the toughest challenges is to develop and conduct qualitative and quantitative evaluation procedures and measurements.

5) Much effort will be devoted toward designing new collaborative projects for teachers and students with reference to their particular social and cultural interests.

CONCLUSIONS

A national project on computers and education in Chile has been presented. The project started with a small prototype and will gradually expand until it reaches at least 100 nodes. The project's assumptions that an attractive and simple to use software platform encourages the meaningful use of computers for stimulating reading and writing and for working on collaborative projects integrating areas of knowledge was partially validated during the prototype. Eventually, the project's final objective is to make public schools a more effective and attractive place for teachers and students regardless of their social, cultural or geographical situation.

REFERENCES

[1] J. J. Hirschbuhl and R. M. Konet (eds.). "Computers in Education", Fourth Edition. The Duskin Publishing Group, 1990. ISBN 0-87967-836-4.

[2] C. Muñoz I. (ed.). "Calidad, equidad y eficiencia de la educación primaria: estado actual de las investigaciones realizadas en américa latina". Centro de Investigación y Desarrollo de la Educación, Santiago, Chile, 1988. ISBN 968-7165-24-3.

[3] J. Sanchez I. "Educational Computing in Chile: Trends, Issues and Recommendations". Educational Technology, March 1991, 54-56.

[4] R. de la Puente and D. Araya. "Estudio sobre equipamiento computacional y su uso en los establecimientos educacionales de la región metropolitana". Ministerio de Educación de Chile, Departamento de Planificación 1991.

[5] O. Park. "Hypermedia: Functional Features and Research Issues". Educational Technology, August 1991, 24-31.

[6] S. Ambron and K. Hooper (eds.) "Learning with Interactive Multimedia. Developing and Using Multimedia Tools in Education". Microsoft Press, 1990, ISBN 1-55615-282-5.

[7] Special issue on developments in educational computing telecommunications. Educational Technology, April 1991.

[8] O.T.A. Report. Superintendent of Documents, Government Printing Office, Washington, D.C. 20402-9325. "Power On! New Tools for Teaching and Learning".

Criterios de Cobertura para Pruebas de Programas Paralelos

Emilio Hernández
emsca!usb!emilio@sun.com
Departamento de Computación y Tecnología de la Información
Universidad Simón Bolívar
Apartado 89000, Caracas 1080-A, Venezuela

Alejandro Teruel
emsca!usb!teruel@sun.com
Departamento de Computación y Tecnología de la Información
Universidad Simón Bolívar
Apartado 89000, Caracas 1080-A, Venezuela

Resumen

En este artículo se propone un esquema de análisis de cobertura para pruebas de programas paralelos con comunicación síncrona. La orientación del criterio definido tiene como objetivo servir de apoyo al desarrollo de software paralelo sobre una plataforma de transputadores.

Keywords: Paralelismo, Transputadores, C, Pruebas de Programas, Complejidad Ciclomática

1 Introducción

La prueba de programas (*testing*) es una fase muy importante del ciclo de desarrollo de software. Con frecuencia el tiempo invertido en esta etapa por algunas organizaciones de elaboración de software se eleva a un 40% del tiempo total de desarrollo [8, pág. 467]. El desarrollo de software paralelo es más difícil que el de software secuencial porque debe controlarse la coordinación de los procesos. Se han elaborado algunas herramientas experimentales de depuración y prueba de programas paralelos [5], de alcances todavía limitados.

En particular, las pruebas de programas secuenciales de tipo caja blanca [7] difícilmente se adaptan a las necesidades de pruebas de programas paralelos, porque deben contemplar elementos externos a la estructura del programa, como son las comunicaciones. En este artículo se propone un criterio de cobertura para hacer pruebas de tipo caja blanca de programas paralelos, en particular para los que utilizan el paradigma MIMD con comunicación síncrona definido por el esquema *OCCAM* de los transputadores INMOS [3]. Este criterio constituye la base para elaborar una herramienta automatizada de ayuda a las pruebas y depuración de programas paralelos.

2 Criterios de cobertura para programas paralelos

2.1 Grafo de Caminos Paralelos

En este trabajo se usó un modelo particular simplificado de un programa paralelo. Este modelo se toma del paradigma *OCCAM* de ejecución de procesos secuenciales en transputadores, cuyos mecanismos de sincronización y comunicación, tomados en cuenta para el análisis que sigue, pueden verse con mayor detalle en [3].

El grafo de caminos de un módulo secuencial contiene un nodo por cada condicional simple (**vértice o nodo de decisión**) y un nodo por cada secuencia maximal de instrucciones no condicionales, tales como asignaciones o llamadas a funciones, que a lo sumo contengan una instrucción de comunicación (**vértice o nodo de acción**). Para esto, los bloques de instrucciones se dividen en subsecuencias que comiencen con *send* o *recv*, excepto posiblemente la primera subsecuencia. Las aristas representan relaciones de precedencia de ejecución entre nodos. Existen dos nodos especiales, *INICIO* y *FIN* que representan los puntos de entrada y salida del módulo. Un camino se entiende en el sentido tradicional que se le da en teoría de grafos, debiendo comenzar en *INICIO* y terminar en *FIN*.

Un grafo de caminos paralelos se define de forma similar al grafo de caminos secuencial, la diferencia es que se trata de un grafo no conexo, en el que cada componente conexa representa a un módulo secuencial. Cada componente conexa i tiene un vértice inicial $INICIO_i$

y un vértice final FIN_i.

2.1.1 Caminos de ejecución paralelos

Dado un módulo paralelo M y su grafo G de caminos asociado a q componentes conexas $G_1 \ldots G_q$, se define como camino de ejecución paralelo una q-upla $(C_1, C_2, \ldots, C_q)$, donde C_i es un camino de ejecución secuencial de la componente conexa G_i del grafo de caminos paralelos. De este modo, el conjunto total de caminos paralelos, que denotaremos como CP_M, es el producto cartesiano de los conjuntos de todos los caminos de los módulos secuenciales. En la figura 1 se muestra un módulo paralelo formado por dos módulos secuenciales y su grafo de caminos, con dos componentes conexas.

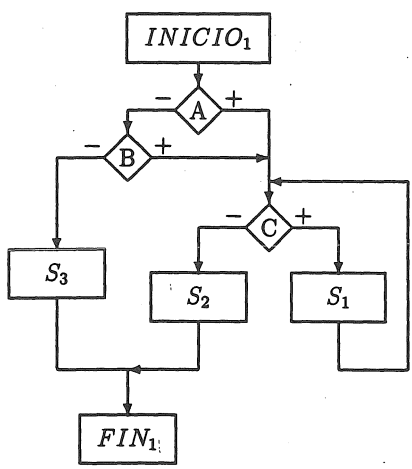
2.1.2 Conjuntos de cobertura paralela

Pueden definirse conjuntos de cobertura paralela de forma similar a los de los módulos secuenciales [7, 8]. Para un módulo paralelo M un conjunto de cobertura es un subconjunto de CP_M que cumple con ciertas condiciones. Se pueden definir varias clases de subconjuntos de CP_M.

- **Conjunto de cobertura paralela exhaustiva** ($CCPE_M$): el mismo CP_M.
- **Conjuntos de cobertura paralela de instrucciones** ($CCPI_M$): subconjuntos de CP_M que cubren todos los vértices de acción, es decir, cada vértice de acción es parte de algún camino de alguna q-upla perteneciente al conjunto.
- **Conjuntos de cobertura paralela de condiciones** ($CCPC_M$): subconjuntos de CP_M que cubren todas las aristas cuya primera componente es un vértice condicional, es decir, cada arista cuya primera componente es un vértice de decisión es parte de algún camino de alguna q-upla perteneciente al conjunto.
- **Conjuntos de cobertura ciclomática independiente** ($CCCicI_M$): subconjuntos de CP_M en los que se tiene una cobertura ciclomática [6] en cada una de las componentes conexas, es decir, en cada una de las dimensiones de las q-uplas del conjunto.

```
M1 () {
   if (A || B) {
      while (C) {
         /* S1 */
         Bloque;
      }
      /* S2 */
      send(M2, msg);
   }
   else {
      /* S3 */
      send(M2, msg);
   }
}
```

```
M2() {
   if (D) {
      /* S4 */
      recv(M1, msg);
   }
   else {
      /* S5 */
      recv(M1, msg)
   }
}
```



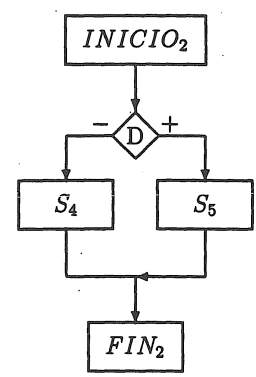


Figura 1: Módulo Paralelo y su Grafo de Caminos

2.2 Grafo de ejecución conjunta

Los conjuntos de cobertura descritos en la sección anterior, excepto $CCPE_M$, no contemplan una cobertura exhaustiva de las posibilidades de comunicación entre los módulos secuenciales. Si bien los tipos de cobertura garantizan la ejecución de todas las operaciones de comunicación al menos una vez, no se tiene un mecanismo que verifique si una instrucción de comunicación en un módulo secuencial se sincroniza exactamente con la que el programador desea, en otro de los módulos. Además, si una instrucción de comunicación puede eventualmente sincronizarse de varias maneras con distintas operaciones similares en otro módulo, no se ofrece hasta ahora ningún tipo de cobertura de tales actos de comunicación entre instrucciones de ambos módulos secuenciales. Por esta razón se ha establecido un esquema de fusión de los grafos de los módulos secuenciales en uno solo, denominado **grafo de ejecución conjunta**, a partir del cual se puede determinar ciertos tipos de cobertura. La elaboración del grafo de ejecución conjunta permite, además, realizar un análisis estático del módulo paralelo (en particular, verificación de compatibilidad de protocolos de comunicación) que puede arrojar información al programador acerca de posibles fuentes de error en el programa. La elaboración un grafo de ejecución conjunta se basa en el hecho de que existe una relación de orden parcial de ejecución entre las instrucciones de todos los módulos secuenciales. Esta relación está determinada por la ejecución de instrucciones de comunicación, que establecen un punto de sincronización entre módulos [4, 1].

El grafo de ejecución conjunta contiene como nodos a todos los que pertenecen a las componentes conexas, cumpliendo las siguientes propiedades:

- Los nodos de *INICIO* y *FIN* de cada componente conexa se combinan en uno sólo de *INICIO* y uno sólo de *FIN*, respectivamente.

- Un nodo perteneciente a una componente conexa puede tener como nodo sucesor a sus sucesores en la misma componente o a un nodo de otra componente conexa, siempre que lo permita la relación de orden parcial que existe en la ejecución de las secuencias de instrucciones representadas por los vértices.

- Todo nodo de comunicación debe estar acoplado (esto es, precedido o seguido) por un nodo de comunicación del módulo y el canal con el que establezca comunicación. La combinación de estos dos nodos de distintos módulos en uno solo dentro del grafo de ejecución conjunta se denomina *nodo de sincronización.*

- Puede haber réplicas de nodos de las componentes conexas en el grafo de ejecución conjunta, siempre que los subgrafos de caminos que continúan a partir de ellos sean distintos. Esto significa que no deben existir, por razones de simplificación, subgrafos idénticos dentro del grafo de ejecución conjunta.

Dado un conjunto de módulos secuenciales que conforman un módulo paralelo, generalmente existen muchos grafos de ejecución conjunta posibles. No obstante, es posible, dado un ordenamiento determinado de los módulos secuenciales, obtener un grafo de ejecución conjunta único, en el que los vértices cuyo orden de ejecución es indeterminado tienen en el grafo una relación de precedencia bien definida [2].

En la figura 2 se da un ejemplo de grafo de ejecución conjunta, construido a partir del módulo paralelo cuyo grafo de caminos independientes se da en la figura 1. En línea punteada están los nodos de sincronización.

3 Conclusiones

Se ha diseñado un criterio de cobertura para programas paralelos que permite realizar análisis estático y dinámico de programas. La construcción del grafo de ejecución conjunta puede dar información acerca de posibles interbloqueos *(deadlocks)*, al encontrarse con la necesidad de acoplar en algún camino dos vértices de comunicación del mismo tipo (*send* o *recv*) pertenecientes a distintos módulos. Por otro lado, una vez construido el grafo de ejecución conjunta puede aplicarse cualquiera de los criterios definidos para cobertura de módulos secuenciales, si consideramos a los vértices de sincronización como un vértice de acción normal y a los vértices repetidos los etiquetamos para diferenciarlos.

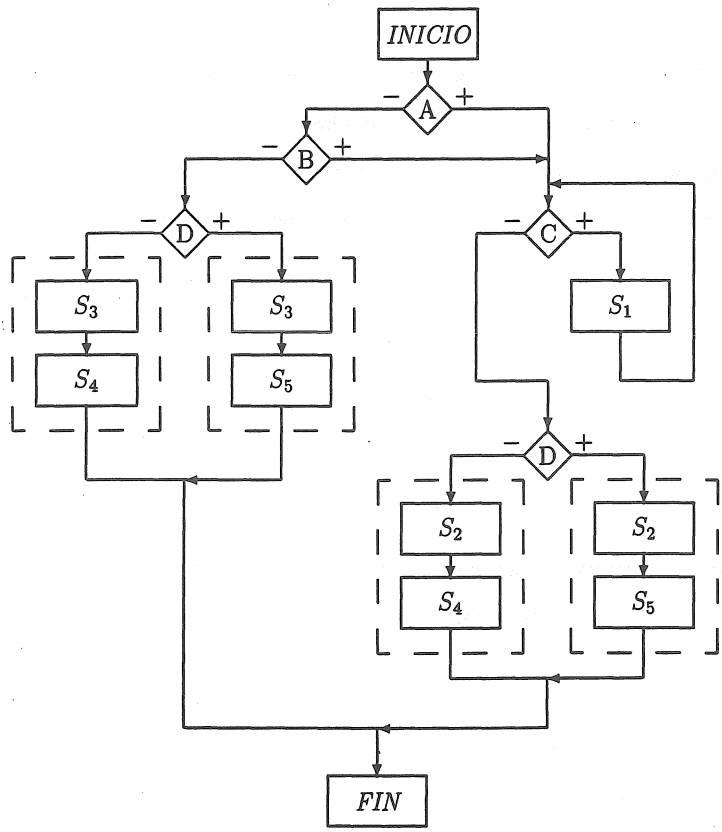


Figura 2: Grafo de ejecución conjunta del módulo de la figura 1

A partir de los criterios de cobertura definidos en este artículo se está diseñando actualmente un prototipo de una herramienta de ayuda a las pruebas de programas paralelos que en base a la instrumentación de los módulos para generar trazas, a la elaboración del grafo de ejecución conjunta, y al análisis *post-mortem* de dichas trazas, se determina el nivel de cobertura de las pruebas realizadas. Desde el punto de vista teórico se están realizando investigaciones para completar el esquema, añadiendo criterios para cobertura de ciclos y buscando simplificaciones cuando el módulo paralelo está formado por réplicas de módulos secuenciales idénticos (caso típico del esquema de programación *master-slave*).

Referencias

[1] C.J. Fidge. Partial orders for parallel debugging. In *Proceedings of the SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging.* SIGPLAN Not. (ACM), 1989.

[2] E. Hernández. Sistema de apoyo a las generación de datos de prueba para programas paralelos. Trabajo de Grado, Maestría en Ciencias de la Computación, Universidad Simón Bolívar, 1992.

[3] INMOS Limited. *Transputer development system*, 1988.

[4] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications ACM*, 21(7), 1978.

[5] M. Lutz. Testing tools. *IEEE Software*, mayo 1990.

[6] T. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4), 1976.

[7] G. Myers. *The Art of Software Testing.* John Wiley and Sons, 1979.

[8] R. Pressman. *Software Engineering.* McGraw-Hill, 1987.

Paralelismo en Visualización Volumétrica

Anita Hung*
emsca!usb!hungleo@sun.com
Centro de Investigación y Soporte Tecnológico de
Petróleos de Venezuela, S.A,
Apartado 76343, Caracas Venezuela 1070

Alejandro Teruel
emsca!usb!teruel@sun.com
Departamento de Computación y Tecnología de la Información
Universidad Simón Bolívar
Apartado 89000, Caracas 1080-A, Venezuela

Resumen

Se estudia la paralelización de un conocido algoritmo de visualización volumétrica sobre una arquitectura MIMD mediante dos esquemas de descomposición de dominios: espacio del objeto y espacio de la imagen. La meta es evaluar el desempeño de cada uno utilizando un esquema de asignación estática de carga de trabajo. Las pruebas fueron realizadas en una plataforma distribuida de estaciones de trabajo SUN, donde la disponibilidad de recursos es exclusiva. Los resultados revelaron que el primer método aventaja considerablemente en efectividad al segundo e indican que en ambos esquemas, factores como el criterio de balance de carga de trabajo y operaciones de manejo de estructuras especiales provenientes de la paralelización, tienen mayor peso que el "overhead" de comunicación.

Keywords: Paralelismo, MIMD, Visualización Volumétrica

1 Introducción

El algoritmo de visualización implementado tiene como base el expuesto por M. Levoy en [10]. Dicho algoritmo utiliza una simplificación del método tradicional de trazado de rayos, en el sentido que obvía los cálculos recursivos debido a las bifurcaciones del rayo por reflexión

*Actualmente en IMG Consultores

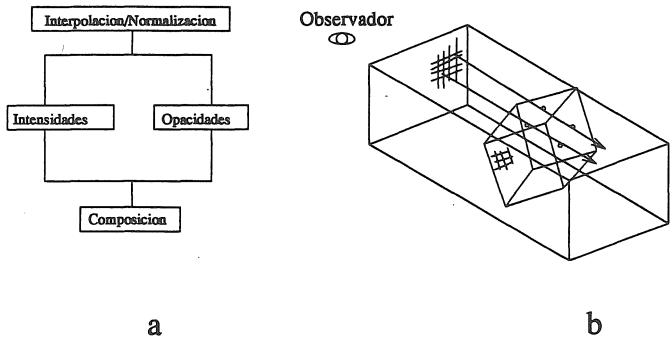


Fig. 1: Método secuencial.

y refracción. Los objetos constituyen los elementos (voxel) de una malla tri-dimensional regular.

Para aplicaciones en el campo de la geofísica, donde el tamaño de los volúmenes a visualizar están en el rango de varios gigabytes, el generar una imagen puede tomar horas, si no días. Frente a esta problemática, el procesamiento paralelo de alto nivel, constituye una solución inmediata. Las idea es distribuir los datos entre los diferentes procesadores procurando que el "overhead" proveniente de las comunicaciones sea mínima.

Algunos trabajos representativos en el área lo constituyen [11], [5], [8] y [2]. Una discusión más completa de los antecedentes se encuentra en [7].

2 Método Secuencial

La figura 1-a resume las principales tareas del método desarrollado. Previo a la visualización del volumen, se aplican una serie de preprocesamientos: a) Interpolación/Normalización de los datos mediante una sencilla interpolación lineal; b) Cálculo de Intensidades-a cada elemento volumétrico se le calcula el color mediante el modelo de Phong [6] c) Cálculo de Opacidades.

La generación de la imagen bi-dimensional se realiza en su totalidad en el módulo de composición. El cálculo del pixel (x_i, y_i) del plano de visualización comienza trazando un rayo, desde el observador, pasando por el pixel en cuestión hacia el volumen (figura 1-b).

Como optimización hemos colocado una condición en donde el rayo detiene su recorrido dentro del volumen cuando la opacidad acumulada sobrepase a uno. El valor final se obtiene de sucesivas composiciones de intensidades y colores de los voxeles que intersecta el rayo. Por otro lado, para reducir los problemas de "aliasing", a cada valor de intensidad y opacidad de dichos voxeles se le aplica un proceso de remuestreo, es decir, se promedia con los respectivos valores de los ocho vecinos más cercanos.

Con el objeto de escoger la metodología de paralelización más conveniente, se realizó un análisis detallado del comportamiento de la versión secuencial. El mismo concluyó que el mayor porcentaje de tiempo en una sola rutina era de alrededor de 50%. Según la llamada "Ley de Amdahl" [1], al aplicar a esta rutina por ejemplo 16 procesadores, no se obtendría un aceleramiento mayor de 2. Tomando en cuenta esta observación y las experiencias de paralelización en otros trabajos como [3] se decidió paralelizar la totalidad del método, lo cual implica distribuir el volumen entre los procesadores.

3 Descomposición del espacio del objeto (De0)

La partición del volumen se realiza seleccionando dos ejes del sistema de coordenadas del objeto, que determinan las directrices u orientación de los planos de corte del volumen. Esta partición origina subvolúmenes que serán posteriormente repartidos entre los procesadores. Para lograr balance de carga, autores como Kobayashi *et al.* y Goldsmith [9], [13], han adoptado un método de asignación de subvolúmenes llamado descomposición por dispersión (*scattering decomposition*) utilizando formatos F. La figura 2-a muestra un ejemplo con $F = 4$ y $P = 9$. En nuestro caso particular, hemos tomado los mismos lineamientos de repartición de carga con la salvedad que los formatos se generan colocando todos los procesadores en una fila, y sucesivas filas se crean haciendo un corrimiento circular hacia la izquierda de la fila inmediata anterior (Fig. 2-b).

Respecto a la arquitectura de paralelización, existen principalmente tres tipos de procesos: a) Generador de rayos (GR), b) Consumidores (C) y c) Colector de pixeles (CP). El nodo GR es el responsable de generar los rayos, intersectarlos con el volumen y enviarlos al nodo C respectivo. Los subvolumenes son distribuidos en los nodos C, quienes calculan

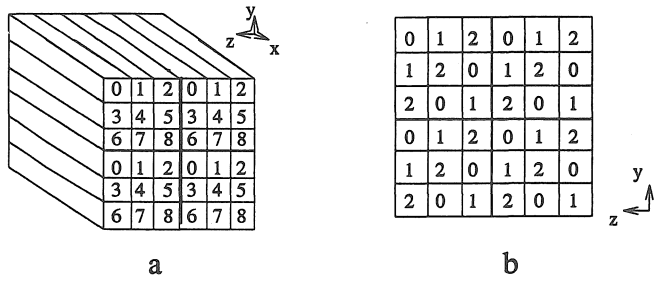


Fig. 2: Descomposición por dispersión.

las interpolaciones, intensidades y opacidades correspondientes. Al finalizar el recorrido, se transmite el resultado al nodo CP. Físicamente, los nodos GR y CP comparten un mismo procesador, mientras que los nodos C se distribuyen en diferentes procesadores. La finalización ocurre cuando el nodo CP recibe todos los resultados, con lo cual transmite la imagen al procesador maestro. El tipo de división adoptado puede obligar que diferentes secciones de un mismo rayo sean trabajados secuencialmente en diferentes procesadores, y por tanto es necesaria una considerable comunicación entre este tipo de nodos.

4 Descomposición del espacio de la imagen (DeI)

En este esquema, la partición está definida por planos de corte que intersectan tanto al volumen como al plano de visualización, y tienen la misma dirección que los proyectores[1]. Tomando la idea de las formatos mencionadas anteriormente, la división del dominio está determinada por el número de procesadores P que intenvendrán en el procesamiento y por el número F de formatos. La figura 3 muestra un ejemplo para $F = 2$ y $P = 3$. La carga de trabajo correspondiente a cada procesador constituye una o varias particiones cuya(s) área del plano de visualización y del volumen se encuentren limitados por el mismo par de planos de corte. Esto implica que cada rayo es procesado en su totalidad en el mismo procesador.

La descomposición comienza seleccionando dos aristas del volumen, paralelas entre sí, quienes junto con el punto del observador determinan la orientación de los planos de corte

[1]Definición tomada de [6]

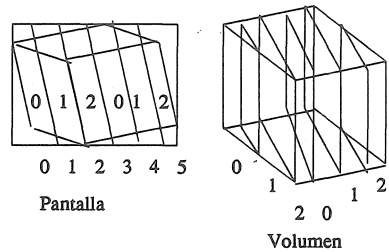


Fig. 3: Particiones de la pantalla y el volumen.

del volumen. Se ha convenido seleccionar aquellas que conforman la silueta de la proyección del volumen. A continuación se calculan $P * F - 1$ rectas paralelas equidistantes entre sí. De esta forma, los planos de corte son aquellos que contienen dichas rectas y que tienen la misma dirección que los proyectores de visualización. Respecto a la arquitectura de paralelización, ésta se simplifica notablemente, pues se traduce en un modelo de servidor y clientes [12].

5 Resultados Experimentales

El código se realizó en lenguaje C. La plataforma consta de nueve estaciones de trabajo SUN/IPX[2] conectadas por una red Ethernet. Con respecto a la comunicación entre los procesos, se utilizó la librería eMP [4]. Para analizar el desempeño se usaron el aceleramiento $A(N,P)$, la fracción serial empríricamente determinado f_c y el desbalance de carga $\delta_{N,P}$, medidos en términos del tiempo $T(N,P)$ empleado en el procesamiento de un problema de tamaño N aplicando P procesadores. Detalles de estas métricas se pueden encontrar en [7].

Las tablas 1, 2 y 3 resumen los resultados obtenidos para los modelos DeO y DeI sobre una red con disponibilidad exclusiva de recursos, en términos de dichas métricas. Los datos utilizados fueron los de un volumen inicial de tamaño $20x20x5$ interpolado a $128x128x128$ y pantalla de $256x256$ pixeles. El nivel de opacidad escogida fué altamente transparente para aumentar al máximo los requerimientos de procesamiento y comunicación.

Haciendo un análisis de dichas tablas, se observa que DeO supera en rendimiento a DeI, a medida que aumenta el número de procesadores. La explicación se debe a una combinación

[2]SUN/IPX es una marca registrada de Sun MicroSystems

	$P=2$	$P=3$	$P=5$	$P=9$
DeO	1.19	2.4652	4.7228	7.2757
DeI	1.91	2.5388	4.0929	5.6756

Tabla 1: Aceleramiento $A(N,P)$

	$P=2$	$P=3$	$P=5$	$P=9$
DeO	0.6807	0.1085	0.0147	0.0296
DeI	0.0471	0.0908	0.0554	0.0732

Tabla 2: Fracción serial f_c

de tres factores:a) Mejor balance de carga en la rutina de composición en DeO que en DeI; b) Menor área de cálculo redundante en DeO que en DeI y c) Estructuras menos costosas de manejar en DeO que en DeI.

Tal parece que el tiempo requerido en la comunicación de rayos en DeO, no juega un papel relevante en el rendimiento general. En este estudio, la arquitectura de paralelización de DeO contempla un complejo manejo de "buffers" y procesos satélites responsables de la transmisión y recepción de mensajes. Se hicieron pruebas con diferentes tamaños de mensajes y cotas superiores sobre el total de rayos procesándose simultáneamente en los nodos tipo C, obteniéndose que dichas variaciones pueden llegar a afectar el aceleramiento, el cual varía entre 5.5 y 7.4 cuando se tienen nueve procesadores. Por otro lado, el diseño de DeO mantiene al proceso GR ocioso cierto tiempo mientras los procesos se ocupan del preprocesamiento del volumen. Lo que quiere decir, que aún con esta desventaja se tiene que DeO tiene mejor rendimiento que DeI. El tiempo ocioso en GR explica también la baja aceleración cuando $C = 1$ en la tabla 1, lo que influye directamente en la fracción serial.

Otro punto interesante que se observa en la tabla 1 es la desaceleración que presentan ambos métodos a medida que aumenta el número de procesadores. En el caso de DeO, si el número de formatos $F > 1$, se obliga a que cada procesador mantenga varios subvolúmenes de los datos, lo cual genera más secciones del rayo lo que incide mayor tráfico en el sistema

	$P=2$	$P=3$	$P=5$	$P=9$
DeO	0.0000	1.2821	1.0526	3.7838
DeI	0.1002	11.332	12.179	17.777

Tabla 3: Desbalance de carga $\delta_{N,P}$

de comunicaciones. Por otro lado, en DeI, el desbalance de carga cobra mayor peso a medida que aumenta el número de procesadores. Obsérvese que el desbalance puede aumentar en casi un 50% al pasar de cinco a nueve procesadores.

Se analizó el rendimiento de DeO en una red no aislada, para ello se escogieron dos niveles de contención de la red a media y alta. Aunque son resultados muy particulares dependiente de las condiciones de la red, indican que DeO puede tener una degeneración de $A(V, F, C)$ de por lo menos un 50%, lo que puede indicar que el rendimiento de DeI puede aventajar a DeO bajo tales condiciones.

6 Conclusiones y Recomendaciones

Es manifiesta la necesidad de paralelizar métodos de visualización volumétrica debido la necesidad de obtener resultados en tiempos razonables. Se han presentado dos esquemas de paralelización para el método de visualización volumétrica basado en [10]. Resultados experimentales indican la factibilidad de tal paralelización.

Entre los dos esquemas de paralelización expuestos, DeO aventaja en rendimiento a DeI, indicando que el tiempo de comunicación no necesariamente puede llegar a dominar el tiempo de procesamiento global en una plataforma distribuida de estaciones de trabajo conectadas mediante una red Ethernet.

La paralelización de métodos de visualización volumétrica requiere poner especial cuidado en los "overhead" de paralelización como son: área de recálculo, facilidad de ubicar un voxel dentro del volumen, balance de carga, comunicación.

La aplicación de técnicas de paralelismo al área de visualización volumétrica ha sido poco explotada y queda mucho por hacer. Este primer prototipo de DeO requiere todavía una serie de afinamientos para lograr tiempos más aceptables de cómputo.

7 Agradecimientos

Deseamos expresar nuestro agradecimiento a las siguientes personas por su valiosa colaboración en el desarrollo de este trabajo: Benjamín Sagalovski, Arturo Puente, Antonio De

Lilla, Angel Sánchez, Lucio Nori, Jorge Manrique, Todd Elvins, José Ramón Jiménez, Jesús Ferrer, Rafael Montaño, Decanato de Postgrado de la Universidad Simón Bolívar y por último pero no menos importante, al Laboratorio de Computación USB.

Referencias

[1] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In AFIPS Press, editor, *AFIPS Conference Proceedings*, pages [483–485], Reston, Va., 1967.

[2] J. Challinger. Parallel volume rendering on a shared-memory multiprocessor. Technical Report UCSC=CRL-91-23, University of California, Santa Cruz, March 1992.

[3] Z. Cvetanovic. The effects of problem partitioning, allocation and granularity on the performance of multiple processor systems. *IEEE Transactions on Computers*, C-36(4):[421–432], April 1987.

[4] A. DeLilla and R. Karia. Implementing eMP an easy message passing library. *Sun User Group Conference Proceedings*, December 1991.

[5] John Ellis, Gershon Kedem, Menon, and Herb Voelcker. Breaking barriers in solid modeling. *Mechanical Engineering*, 113(2):[28–34], February 1991.

[6] J. D. Foley, A. Van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice.* Addison Wesley, 1990.

[7] A. Hung. Composición volumétrica en paralelo. Master's thesis, Universidad Simón Bolívar, 1991.

[8] A. Kaufman and R. Bakalash. Memory and processor architecture for 3d voxel based imagery. 8(11):[10–23], November 1988.

[9] H. Kobayashi and H. Kubota and Y. Shigei S. Nishimura. Load balancing strategies for a ray parallel ray-tracing system based on constant subdivision. *The Visual Computer*, pages [197–209], April 1988.

[10] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[11] Marc Levoy. Design for a real-time high-quality volume rendering workstation. In *Chapel Hill Workshop on Volume Visualization Conference Proceedings*, pages [85–92], 1989.

[12] David May and Roger Shepherd. Communicating process computers. Technical report, INMOS Ltd., Bristol, 1987. Inmos Technical Note 22.

[13] J. Salmon and J. Goldsmith. A ray tracer hypercube. *Communications of the ACM*, pages [1194–1286], 1988.

PROBLEMA DE SCHEDULING RESUELTO MEDIANTE UN SISTEMA DE RESTRICCIONES TEMPORALES

Ibañez, F. *[A],[B]*
Forradellas, R *[A],[B]*
Toledo, F *[A],[B]*
Berlanga, R. *[A]*
Barber, F. *[A]*
Martin, G. *[A]*

[A] LISITT- Universidad de Valencia
C/Dr.Moliner 50 -46100 Burjassot -Valencia- España
FAX: +34-6-386 4568 Email:LISITT@EVALUN11.BITNET

[B] Universitat Jaume I
Campus de Penyeta Roja-Castellón-España
FAX: +34-64-345842

RESUMEN:

Partiendo de un problema de scheduling [4][5][6], el que puede expresarse en términos generales como tareas que deben ser procesadas utilizando determinados recursos, se presenta una solución a dicho problema mediante un sistema de restricciones temporales, el cual será descrito en el presente trabajo.

El sistema propuesto es una herramienta basada en CLP(Constraint Logic Programming) que brinda un entorno integrado para manejar restricciones temporales, las que pueden ser representadas básicamente de dos maneras: numérica y simbólica. En la mayoría de las aplicaciones reales, tanto las cantidades involucradas en las restricciones numéricas, como la información expresable mediante restricciones simbólicas, no están totalmente definidas a priori. El presente trabajo permite representar esta incertidumbre y reducirla como consecuencia de las restricciones definidas.

El trabajo incluye un análisis de diferentes métodos de optimización y su resolución mediante el sistema propuesto.

Palabras Clave: Scheduling, razonamiento basado en restricciones y razonamiento temporal.

Este trabajo fue parcialmente financiado por el Proyecto de la CEE Esprit 5291 "CHIC"

1. INTRODUCCIÓN

Se entiende como problema de scheduling el que surge cuando se busca una secuencia óptima de tareas que deben procesarse con un número finito de recursos. Esta clase de problemas pueden considerarse como combinatorios discretos ya que, al igual que otros problemas surgidos en áreas de aplicación de la IA, tienen como característica la búsqueda en un espacio finito y discreto de una solución que satisfaga un conjunto de restricciones fijadas a priori. En estos problemas hay que enfrentarse con una complejidad alta (generalmente NP-completitud), lo cual implica la no existencia de un algoritmo general y eficiente para resolverlos.

Habitualmente el medio más utilizado para la resolución de los problemas combinatorios discretos viene siendo la realización de programas especializados escritos en lenguajes procedurales. Estos programas son generalmente muy eficientes pero poco flexibles.

La programación lógica permite expresar los problemas combinatorios discretos debido a su no determinismo, su forma relacional y su semántica declarativa. Sin embargo, los métodos de resolución existentes en programación lógica para estos problemas son muy ineficientes [1][3].

La inclusión de técnicas de consistencia dentro de la programación lógica mejora substancialmente la eficiencia para la resolución de los problemas que estamos tratando [1]. La combinación de la programación lógica y de técnicas de consistencia ha dado lugar a los nuevos lenguajes lógicos denominados CLP (Constraint Logic Programming). Aunque estos lenguajes no llegan a alcanzar los niveles de eficiencia de los programas procedurales, sí proporcionan niveles bastante aceptables (de varios órdenes por encima de la programación lógica sin restricciones). Esta pérdida mínima de eficiencia viene compensada con la declaratividad ganada. Actualmente existen numerosos compiladores Prolog que incluyen el manejo de restricciones y añaden nuevos dominios de computación como los racionales, booleanos, dominios finitos naturales, enteros, etc... , algunos de éstos son CHIP, Prolog III, Pecos, etc ...

Dado el carácter temporal de los problemas de scheduling, algunos de estos pueden ser expresados mediante los diferentes modelos temporales desarrollados para el manejo de eventos temporales y sus relaciones. De todos los formalismos existentes destaca el modelo desarrollado por Detcher, Meiri y Pearl, ya que trabaja con información temporal métrica y el mantenimiento de la consistencia se realiza mediante técnicas CSP (Constraint Solving Problems) que le proporcionan muy buena eficiencia [12]. Desafortunadamente, este formalismo no puede expresar cualquier tipo de información temporal, ni siquiera disyunciones a nivel de intervalos que tan frecuentemente aparecen en problemas de scheduling. Además, al igual que el resto de formalismos, la obtención de soluciones suele ser bastante costosa debido a que como norma general el mantenimiento de la consistencia es parcial.

Por estas razones los modelos de razonamiento temporal no son frecuentes para la resolución de problemas de scheduling y tienen una aplicación muy limitada. De acuerdo con lo anterior CHIP, [2][9] es un lenguaje cuya eficiencia debe ser investigada para estas cuestiones.

2. SISTEMA DE RESTRICCIONES TEMPORALES

El modelo de razonamiento temporal que se describe a continuación se basa en la representación de información temporal de distintos formalismos y pretende ser una integración natural de los mismos. El mantenimiento de la consistencia, la obtención de soluciones y el añadido de nueva información (update) se realiza mediante restricciones sobre variables de dominio finito sobre naturales [1][2][9].

2.1. Formalismo

El sistema admite dos tipos de objetos temporales:

Puntos deslocalizados: Representan magnitudes cuya localización es incierta dentro de un dominio dado a priori. Estos son representados en el lenguaje CHIP mediante variables de dominio finito.

Se declaran mediante el operador predefinido ´::´, con la sintaxis siguiente:

X:: Min:Max % X es una variable cuyo dominio es el intervalo [Min,Max].

Si tenemos un conjunto de variables dominio que inicialmente parten con un mismo dominio, entonces podemos declararlas mediante una lista:

[X1,X2, ... ,Xn] :: Min : Max % las variables de la lista toman sus valores en [Min,Max]

Nótese que en relación con otros formalismos, los puntos deslocalizados pueden representar los eventos puntuales de los modelos de Kautz y Vilain [8] y Dechter, Meiri y Pearl [12].

Intervalos deslocalizados: Representan eventos temporales con cierta duración. Estos se representan en CHIP mediante una estructura de tres variables dominio que representarían el inicio, final y longitud del intervalo, opcionalmente, cualquiera de estas componentes puede ser constante.

En el sistema propuesto, los intervalos deslocalizados se declaran mediante el operador ´:>´ con la sintaxis siguiente:

nombre :> [Min,Max,Length]

donde, *´nombre´* es un átomo que actua como identificador del intervalo, *Min* y *Max* son dos números naturales que delimitan el espacio de ocupación del intervalo deslocalizado, es decir, los márgenes dentro de los cuales puede localizarse el intervalo. *'Length'* es la longitud del intervalo, que puede ser bien una constante o una variable dominio previamente declarada, bien un functor de la forma *up(M,N)* que permite definir la longitud del intervalo como un punto deslocalizado de dominio *[M,N]*, siendo *M* y *N* dos constantes naturales, sin una declaración previa.

Para acceder a los campos de un intervalo deslocalizado genérico *A* se definen las funciones *Start(A), End(A) y Length(A)*, que devuelven puntos deslocalizados que representan el inicio, final y longitud del intervalo *A*. Internamente, el sistema creará la restricción métrica que relaciona los bordes del intervalo con su longitud: *Start(A) + Length(A) #= End(A).*

Nótese que los intervalos deslocalizados aquí definidos corresponderían con los intervalos del sistema propuesto por Allen [7], aunque en éste los intervalos no pueden expresarse de forma métrica ya que sólo admite relaciones de tipo simbólico. La representación de intervalos temporales en los modelos basados en puntos es incompleta puesto que las disyunciones a nivel de intervalos no pueden ser expresados con esos modelos. De hecho este es uno de los

problemas constantes en razonamiento temporal: la integración de las diferentes representaciones temporales [10][11].

En el Sistema que se propone, todas las relaciones entre eventos temporales, sean intervalo-intervalo, punto-punto o punto-intervalo, se transforman internamente en restricciones lineales predefinidas en CHIP. La resolución de estas restricciones se realiza a través de la regla de inferencia PLAIR (Partial Looking Ahead), la cual produce la poda parcial de valores inconsistentes de las variables dominio implicadas en la restricción (reducción de dominios) y propaga estas reducciones a todas las restricciones que contengan alguna de las variables cuyo dominio se haya reducido (restricciones activas). Un amplio tratamiento de esta regla de inferencia y otras definidas en CHIP puede encontrarse en [1] y [9].

El usuario puede manejar directamente los objetos temporales y establecer sus relaciones mediante un lenguaje superior. Este lenguaje tiene varias partes: declaración de objetos temporales, vista anteriormente, restricciones temporales, consulta y obtención de soluciones.

2.2. Tipos de restricciones

Métricas:

Estas restricciones actuan siempre a nivel de puntos y son representadas mediante restricciones predefinidas en el lenguaje CHIP con la sintaxis siguiente:

<TL_1> <operador> <TL_2>

donde *TL_1* y *TL_2* representan términos lineales sobre variables dominio o puntos deslocalizados definidos como en [2].

<operador> es uno de los operadores de restricción predefinidos en CHIP: #< (menor), #> (mayor), #= (igual), #>= (mayor o igual), #<= (menor o igual) y ## (distinto).

Simbólicas a nivel de intervalos:

Estas restricciones actúan exclusivamente sobre intervalos deslocalizados y su sintaxis viene dada por:

<identificador_intervalo_1> {<relaciones>} <identificador_intervalo_2>

donde los miembros derecho e izquierdo de la expresión son los identificadores de dos intervalos deslocalizados previamente decalarados. El operador <relaciones> es un subconjunto de las relaciones simbólicas de Allen [7] las cuales conformarán una restricción disyuntiva en el caso en que dicho subconjunto tenga más de un elemento.

Ejemplo: La intersección temporal entre el periodo de comida y trabajo vendría expresado por:

```
comer :>[3,10,up(1,2)],
trabajar :>[3,10,up(2,4)],          %declaración de intervalos
comer {overlaps} trabajar,          %restricción simbólica no disyuntiva
End(comer) #= Start(trabajar) + 5   %restrición métrica
```

2.3. Obtención de soluciones

La obtención de soluciones en el sistema descansa en las potencialidades de CHIP, que asegura la completitud del sistema mediante la instanciación de las variables [1]. Contrariamente a otros sistemas basados en red [12], la obtención de soluciones en CHIP es muy eficiente debido a que las restricciones siempre permanecen activas gracias a las reglas de inferencia PLAIR y Forward Checking [1][9]. La obtención de valores de una variable dominio X genérica, se realiza mediante el predicado predefinido ***indomain(X)***. Este predicado es no de-

terminístico e instancia la variable *X* a cada uno de los valores de su dominio que satisfaga todas las restricciones donde participe la variable. Debemos hacer notar que los valores obtenidos a través de *indomain* puede que no pertenezcan a ninguna solución final debido a la parcialidad del mantenimiento de la consistencia en CHIP. La forma de obtener soluciones finales a partir de un conjunto de variables dominio *X1, ..., Xn* se realiza mediante el siguiente predicado:

```
labeling([]).
labeling([X|T]):-
        indomain(X),
        labeling(T).
```

Este predicado genera juegos de valores de las variables dominio pasadas en forma de lista, que satisfacen todas las restricciones, es decir, genera soluciones finales.

En el sistema propuesto, la obtención de soluciones a nivel de intervalos se realiza generando soluciones para las variables dominio que representan el comienzo y la longitud de cada intervalo. El predicado que realiza la generación de soluciones para intervalos se denomina *instanciate/1*, al cual se le pasa como parámetro una lista de los identificadores de los intervalos involucrados.

3. EL PROBLEMA GENERAL DE SCHEDULING

La terminología de la teoría de scheduling surge en numerosas aplicaciones industriales, es por ello que nos referiremos a *tareas* y a *máquinas* para nombrar a cualquier actividad y a cualquier procesador respectivamente [4]. En el planteamiento inicial del problema debemos conocer el número de tareas y el número de máquinas que permitirán realizar estas tareas.

3.1. Definiciones

Entendemos como **operación** al procesamiento de una tarea sobre una máquina. Denotaremos con $\mathbf{o_{ij}}$ a la operación de la i-ésima tarea ejecutada por la j-ésima máquina. Denotaremos con $\mathbf{p_{ij}}$ al tiempo que tarda en ejecutarse la operación $\mathbf{o_{ij}}$.

Las operaciones correspondientes a una tarea deben procesarse en un determinado orden, las restricciones que definen dicho orden son conocidas como *Restricciones Tecnológicas* (RT).

Consideraremos que las tareas están listas para ejecutarse a partir de un determinado tiempo, denominado **'ready time'**. Denotaremos con $\mathbf{r_i}$ al tiempo en el cual la tarea i-ésima está lista para ser procesada.

El problema general de scheduling es encontrar una secuencia de uso de las máquinas para cada cada tarea que sea *compatible* con las RT y *óptima* con respecto a algún criterio de performance.

3.2. Hipótesis

Enumeremos el conjunto de hipótesis de partida que se imponen generalmente en el planteamiento de un problema de scheduling:

1) No pueden procesarse simultaneamente ningún par de operaciones de una misma tarea.
2) Cada operación, una vez comenzada, no puede ser interrumpida.
3) Cada tarea tiene *m* operaciones distintas, una sobre cada máquina. Esto implica que un mismo trabajo no puede realizar más de un procesamiento sobre la misma máquina durante

el tiempo que dura el scheduling. Por otro lado implica que cada tarea utiliza todas las máquinas.
4) Los tiempos de procesamiento son independientes del scheduling. Esta hipótesis implica por un lado que el tiempo para preparar la máquina para su procesamiento es independiente de la última tarea procesada y por otro lado que los tiempos para mover las tareas entre las máquinas son despreciables.
5) Las tareas pueden esperar hasta que las máquinas estén disponibles.
6) Ninguna máquina puede procesar más de una operación al mismo tiempo.
7) Las máquinas nunca fallan y están siempre disponibles a través del periodo de scheduling.
8) Las RT son conocidas a priori y no pueden cambiar.
9) No existe incertidumbre. En particular el número de tareas y máquinas, los tiempos de procesamiento y los ready times son conocidos y fijos.

4. RESOLUCIÓN DEL PROBLEMA.

Supongamos, sin pérdida de generalidad, que tenemos dos tareas y tres máquinas. Las RT definen para cada tarea la secuencia en que serán utilizadas las máquinas. Por ejemplo, las RT:

Tarea 1: o_{12} o_{13} o_{11}
Tarea 2: o_{21} o_{23} o_{22}

implican que la tarea 1 utilizará primero la maquina 2, luego la 3 y finalmente la 1, mientras que la tarea 2 utilizará las máquinas en la secuencia 1, 3, 2.

Como paso inicial, expresaremos cada una de las operaciones o_{ij} como intervalos que en principio ocupan todo el espacio de soluciones del scheduling.

scheduling :-
o11:>[r1,n,p11], o12:>[r1,n,p12], o13:>[r1,n,p13],
o21:>[r2,n,p21], o22:>[r2,n,p22], o23:>[r2,n,p23)],

donde *n* es una constante suficientemente grande que actúa como cota superior de todo el scheduling, *r1* y *r2* son los 'ready time' de las tareas y p*ij* son los tiempos de procesamiento, definidos como en el apartado anterior.

Las RT se representan añadiendo las restricciones siguientes:

End(o12)#<=Start(o13), End(o13)#<=Start(o11), % para la tarea 1
End(o21)#<=Start(o23), End(o23)#<=Start(o22), % para la tarea 2

Luego se incluyen las restricciones disyuntivas para expresar el hecho de que una máquina no puede ser utilizada simultáneamente por dos operaciones (hipótesis 6). En nuestro caso:

o11 {before, after} o21,
o12 {before, after} o22,
o13 {before, after} o23,

Finalmente, se incluye una variable que indique el tiempo de finalización del scheduling del siguiente modo:

End_sch::0:n,
End(o11)#<= End_sch,
End(o22)#<= End_sch,

Hacemos notar que estas restricciones afectan solamente a las últimas operaciones de cada tarea.

La utilización del predicado predefinido *instanciate([o11,o12,o13,o21,o22,o23])* generará todos los schedulings factibles, es decir, que cumplen todas las restricciones.

5. TRATAMIENTO DE HIPÓTESIS MENOS RESTRICTIVAS.

Reconsideraremos las hipótesis descritas en la sección 3.1 con el objeto de dar mayor flexibilidad al tratamiento de scheduling.

Hipótesis 1: Si en una determinada aplicación necesitáramos permitir que un par de operaciones de una misma tarea (por ejemplo: *o12* y *o13*) puedan ejecutarse simultáneamente, sólo debemos excluir del programa anterior la restricción: *End(o12)#<=Start(o13)*. Si quisiéramos expresar además que la operación *o13* puede empezar hasta 2 unidades de tiempo antes de que finalice la operación *o12*, incluiríamos la restricción: *End(o12)#<=Start(o13)+2*.

Hipótesis 2: Si necesitáramos descomponer una operación de una duración determinada ,o incluso indeterminada dentro de ciertos límites, en dos o más suboperaciones, sólo debemos agregar a las relaciones temporales deseadas para estas suboperaciones, una restricción que relacione las longitudes de las mismas. Por ejemplo, si la operación o11, de duración 20, debe descomponerse en las suboperaciones *o111* y *o112*, se debe incluir la restricción: *Length(o111)+Length(o112)#=20*.

Hipótesis 3: Si necesitáramos que una tarea realice más de una operación sobre una misma máquina, debemos realizar dos cambios en el programa general para reflejarlo; por un lado, especificar las relaciones temporales de cada una de las nuevas operaciones, y por otro, incluir nuevas restricciones disyuntivas de éstas. En el ejemplo, si después de la operación *o11* de la tarea 1, necesitáramos realizar otra operación sobre la máquina 3, sólo debemos agregar en el programa: *End(o11)#<=Start(otra_o13) , otra_o13{before,after}o23*.

Por otro lado, si abandonamos la hipótesis de suponer que todas las tareas ocupan todas las máquinas, la solución matemática es substancialmente diferente [4]. Sin embargo la estructura del programa, usando el sistema propuesto, es exactamente la misma. Por ejempo, si la tarea 1 no usara la máquina 3, sólo debemos excluir la declacración del intervalo *o13* y todas las restricciones en que estaba involucrado.

Aunque podría pensarse que una tarea que no usa una máquina podría ser implementada asignándole un tiempo de procesamiento igual a cero a la operación correspondiente a esa máquina, todavía tendríamos un problema: ¿Dónde serían ubicadas estas operaciones nulas en la secuencia de procesamiento de la tarea? A partir de la hipótesis 2 inicial, la tarea estaría esperando que se torne disponible una máquina que en realidad no usa.

Hipótesis 4: Supongamos que el tiempo de procesamiento de la operación *o11* es de 10 o de 13, dependiendo de que esta operación se realice antes o después de la operación *o21* (que es la otra operación que usa la misma máquina). El cambio necesario en el programa general puede expresarse declarando el intervalo correspondiente a la operación *o11* con duración inderminada entre los valores 10 y 13 y expresando las dos posibilidades mencionadas como 'puntos de elección' del siguiente modo:

o11:>[r1,n,up(10,13)],
(End(o11)#<=Start(o21), Length(o11)#=10 ;
End(o21)#<=Start(o11), Length(o11)#=13),

Sin embargo, el tratamiento necesario para casos más generales se torna más complejo e ineficiente.

Hipotesis 5: Supongamos que en una determinada aplicación, necesitáramos comenzar la operación *o13*, a lo sumo 5 unidades de tiempo después de finalizada la operación *o12*, la única restricción que debe incluirse en el programa es: *Start(o13) #<=End(o12)+5*.

Esta es una hipótesis no trivial. En algunos problemas el procesamiento de las tareas debe ser continuo de una operación a la otra.

Algunos casos particulares de las hipótesis no consideradas pueden resolverse en el sistema propuesto pero no son generalizables y excede el alcance de este artículo.

6. OPTIMIZACIÓN

En este apartado analizaremos los criterios de optimización usualmente planteados en scheduling, y mostraremos como son resueltos en el sistema propuesto. Existen muchos más criterios que los aquí descritos, pero generalmente estos se aplican a casos muy particulares y de difícil resolución, es por ello que no serán tratados en el presente artículo.

6.1. Minimización del tiempo de finalización del scheduling

Para cada tarea se define el tiempo de su finalización, denotado por C_i, y a partir de éstos definimos el tiempo de finalización del scheduling como $C_{máx}$ = *máximo*$\{C_i\}$ y el tiempo medio de finalización del scheduling C como la media aritmética de los C_i.

La variable *End_sch*, que representa $C_{máx}$, quedará reducida como consecuencia de las declaraciones y restricciones previas. Un intento razonable para minimizar $C_{máx}$ sería tomar el mínimo elemento del dominio al cual se redujo *End_sch*. Un scheduling óptimo puede encontrarse asignando a los comienzos de cada intervalo el valor más pequeño. Sin embargo este método sólo es válido para un scheduling sin disyunciones y con tiempos de procesamiento conocidos y fijos [1].

Un modo de completar la solución óptima válida para todos los casos es agregar al programa anterior:

indomain(End_sch),
instanciate([o11,o12,o13,o21,o22,o23]).

el cual obtiene como primera solución la óptima. Notar que la instanciación de los intervalos podría hacer fallar las restricciones y necesitar considerar el segundo valor de la variable *End_sch*, como consecuencia de *indomain(End_sch)*.

6.2. Minimización del flujo máximo.

Se define el flujo de la tarea i-ésima, denotado por F_i, como la diferencia entre el tiempo de finalización de la tarea y su 'ready time'. Se define el flujo máximo del scheduling como el máximo de todos los flujos de las tareas: $F_{máx}$ = *máximo*$\{F_i\}$.

En el ejemplo: *F1=End(o11) - r1* , *F2=End(o22) - r2*

El criterio que seguimos para obtener la solución óptima es similar al apartado anterior pero aplicado a la variable $F_{máx}$:

[F1,F2,Fmax] :: 0 : n, *%declaración de los flujos*
F1+r1 #= End(o11),

F2+r2 #= End(o22),
F1 #<= Fmax, F2 #<= Fmax,
indomain(Fmax), instanciate([*o11,o12,o13,o21,o22,o23*]).

6.3. Minimización del flujo medio

Definimos flujo medio **F** como la media aritmética de los flujos de cada tarea. La solución que minimiza **F** es la misma que minimiza **C** [4] (aunque **F** y **C** sean diferentes).

En el ejemplo, debemos minimizar *(C1+C2)/2* que es lo mismo que minimizar *End(o11)* + *End(o12)* lo cual puede expresarse como:

C :: 0 : n,
C#=End(o11)+End(o12),
indomain(C), instanciate([*o11,o12,o13,o21,o22,o23*]).

6.4. Minimización de promedios pesados

En general deseamos minimizar una expresión de la forma $\Sigma\alpha_i\ C_i$, donde α_i es una constante natural que representa el peso asociado a la finalización de la tarea i-ésima C_i.

En nuestro caso $\alpha 1\ C1+\alpha 2\ C2 = \alpha 1\ End(o11) + \alpha 2\ End(o22)$, por lo tanto escribiremos:

P :: 0 : n,
*P #= α1*End(o11)+α2 * End(o22),*
indomain(P), instance([*o11,o12,o13,o21,o22,o23*]).

6.5.Minimización del retraso máximo

Se define el *'due time'* de la tarea i-ésima, denotado por $\mathbf{d_i}$, como el tiempo en el cual debe estar finalizada esa tarea.

Se define el *retraso* de la tarea i-ésima, denotado por $\mathbf{L_i}$, como la diferencia entre la finalización de una tarea y su 'due time'.

En el ejemplo $L_1 = (End(o_{11}) - d_1)$ y $L_2 = (End(o_{22}) - d_2)$. Tanto L_1 como L_2 pueden resultar negativos para el caso en que la tarea finalice antes del 'due time'. Para estos casos, puesto que CHIP trabaja con variables dominio sobre naturales, sumaremos una constante *c* suficientemente grande de manera que obtengamos un retraso "desplazado" positivo:

L1despl= (End(o11) - d1) + c, *L2despl= (End(o22) - d2) + c*

Puesto que minimizar *Lmax* es lo mismo que minimizar *Lmax* desplazado, la solución óptima la podemos obtener de un modo similar a los programas anteriores:

[L1despl,L2despl,Lmax_despl] :: 0 : n,
L1despl+d1#=End(o11)+ c,
L2despl+d2#=End(o22)+ c,
L1despl#<=Lmax_despl,
L2despl#<=Lmax_despl,
indomain(Lmax_despl), instance([*o11,o12,o13,o21,o22,o23*]).

7. CONCLUSIONES

CHIP presenta ciertas ventajas en términos de declaratividad y proporciona una eficiencia cercana a la lograda con programas de propósitos particulares, escritos en lenguajes procedurales [1][3]. Sin embargo, existe una pérdida de expresividad en el manejo de información

temporal debido a la transformación que hay que realizar sobre las relaciones temporales, para expresarlas en restricciones predefinidas en CHIP. En el sistema propuesto el usuario declara los objetos temporales y especifica las relaciones entre los mismos, realizando el sistema las transformaciones necesarias.

El principal problema de eficiencia surge con el tratamiento de las disyunciones que son expresadas y tratadas mediante puntos de elección dentro de un programa lógico. Este método es ampliamente tratado por P. Van Hentenryck en [1] y consiste básicamente en hacer uso del no determinismo de Prolog para crear un punto de elección que permita evaluar una tras otra las restricciones de la disyunción. El principal inconveniente de este método es el elevado costo que supone la evaluación de cada disyunción, ya que cada vez que se evalúa una nueva restricción de la disyunción, se deben restaurar los valores de todas las variables afectadas por la restricción anterioriormente evaluada. Nótese además que las disyunciones así tratadas, no permanecen activas en su conjunto, sino que sólo estarán disponibles en el momento de obtener soluciones. Este problema está parcialmente resuelto en nuestro enfoque mediante el tratamiento de disyunciones a nivel de variables dominio (ver apéndice). Sin embargo, las disyunciones que involucren a más de dos intervalos, seguirán siendo tratadas mediante el método del punto de elección.

En cuanto a las aplicaciones, existe una extensa literatura donde se muestran distintos sistemas reales expresados en el modelo presentado [4][5][6]. Sin embargo, no existe un sistema flexible unificado y eficiente para representar los distintos casos particulares de este modelo. Además, la complejidad matemática varía considerablemente para cada caso particular. El presente trabajo está orientado a la representación de este modelo en un sistema que permita expresar los distintos casos de un modo unificado y susceptible a modificaciones.

AGRADECIMIENTOS

Parte de la investigación realizada en el presente trabajo ha sido realizada mientras los autores trabajaban en el Proyecto Esprit 5291 de la CEE "CHIC" (Constraint Handling in Industry and Commerce). Organizaciones colaboradoras son: ECRC y Siemens (Alemania), ICL e Imperial College (Gran Bretaña), Bull, Reanault, CERT/ONERA y Dassault (Francia), Braghenti y AIS (Italia), CMSU (Grecia) y OCT (España).

APÉNDICE

El método que desarrollamos consiste básicamente en expresar las disyunciones entre dos intervalos a nivel de las variables dominio, en vez de expresarlo a nivel de restricciones. De este modo, se eliminan los puntos de elección y las disyunciones permanecen en todo momento activas.

Para realizar la disyunción de variables dominio es preciso definir una nueva restricción denominada *unión de variables dominio* cuya sintaxis sería:.

$$D = D1 \text{ U } D2 \text{ U } ..... \text{ U } Dn$$

donde *D, D1, D2, ... , Dn* son variables dominio sobre naturales.

Semánticamente, *D* toma como dominio la unión de los dominios de las variables *D1,..,Dn*. Esta restricción hace uso de la regla de inferencia PLAIR.

A modo de ejemplo, sean las variables dominio *D1*::0:3, *D2*::0:5 y *D3*::7:9, sobre las cuales imponemos la restricción

$$D = D1 \text{ U } D2 \text{ U } D3 \text{ (I)},$$

entonces, D tomará sus valores en el dominio definido por la unión de los dominios de *D1*, *D2* y *D3*, esto es *D*=[0,5] U [7,9]. Supongamos ahora que se produce una reducción en la variable *D*, por ejemplo introduciendo la restricción unaria *D#<8*, ello provocará una propagación hacia la restricción (I) que producirá una reducción en el dominio de D3 únicamente, *D3*=[7,8], el resto de variables no se ven afectadas. Si hubiesemos introducido la restricción *D#>7*, entonces las variables *D1* y *D2* se desconectarían de la unión.

Como paso previo a la transformación de las disyunciones entre intervalos se debe realizar una

transformación de todas las relaciones simbólicas implicadas en la disyunción a sus respectivas métricas. Para esta transformación se realiza un mapa de distancias entre los dos intervalos implicados y se extraen las restricciones que deben cumplir las distancias L1, L2, LA y LB (ver figura 1). Todas las relaciones simbólicas cumplirán las siguientes restricciones:

$$Start(B) \#= L1 + End(A) \quad \text{(R-I)}$$
$$End(B) \#= L2 + Start(A) \quad \text{(R-II)}$$

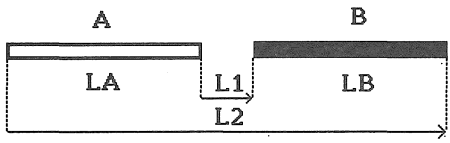


Fig 1. Mapa de distancias entre dos intervalos

A modo de ejemplo obtendremos la transformación de la restricción simbólica *A {before} B* (figura 1) en sus correspondientes métricas.

En este caso tanto *L1* como *L2* son variables de dominio positivo. La restricción de distancias que define esta relación es:

$$LA + L1 + LB \#= L2$$

donde LA y LB son las longitudes de los intervalos A y B respectivamente.

Para todas las relaciones simbólicas existe un conjunto de restricciones sobre las distancias indicadas, que conforman su transformación métrica.

Para cada relación simbólica *i*-ésima de la disyunción obtendremos un juego de variables dominio L1*i*, L2*i*, LA*i* y LB*i* y un conjunto de restricciones métricas sobre éstas. Las retricciones (R-I) y (R-II) son comunes a toda la disyunción, definiendo L1 como la unión de los casos particulares L1*i* y L2 como la unión de las variables L2*i*. Dado que las longitudes de los intervalos A y B están condicionadas a los casos particulares de la disyunción, deberemos de crear dos variables unión, LA y LB que contengan los casos particulares LA*i* y LB*i*. Después deberemos igualar las longitudes de los intervalos con las correspondientes uniones:

$$Lenght(A) \#= LA,$$
$$Lenght(B) \#= LB$$

A la hora de implementar este método en CHIP nos encontramos con el problema de que sólo admite dominios naturales, pero en varias relaciones simbólicas L1 y L2 son dominios de valores negativos. Este problema se soluciona desplazando todas las variables dominio de la unión cierta cantidad para que el dominio pase a ser positivo. Esa cantidad desplazada deberá estar expresada en las restricciones (R-I) y (R-II) de modo que no se altere el resultado.

BIBLIOGRAFÍA

[1] P. Van Hentenryck. "Constraints Satisfaction in Logic Programming" Logic Programming Series. MIT PRESS, Cambridge, 1989.

[2] COSYTEC S.A. "CHIP USER´s Guide". (COSY/REF/001,V 1.1), October 1991.

[3] M.Dincbas, H.Simonis, P. Van Hentenryck. "Solving Large Scheduling Problems in Logic Programming" EURO-TIMS Joint International Conference on Operations Research and Management Science, Paris, France, Julio 1988.

[4] S.French. "Sequencing and Scheduling" Ellis Horwood Ltd, 1982.

[5] R.Bellman, A.O. Esogbue, I. Nabeshima "Mathematical aspects of scheduling and applications" Pergamon Press, 1982.

[6] R.W Conway, W.L Maxwell, L W Miller "Theory of scheduling" Addison Wesley, 1967.

[7] J.Allen "Towards a general theory of action and time" Artificial Intelligence, 23:123-153, 1984.

[8] M.Vilain and H.Kautz "Constraint propagation algorithms for temporal reasoning" Proceedings of AAAI-86, 377-382, 1986.

[9] R.Forradellas, F.Ibañez, R.Berlanga y G.Martín "Manejo de constraints en CHIP", PRODE-91, 234-244, 1991.

[10] I.Meiri "Combining qualitative and quantitaive constraints in temporal reasoning" Proceedings AAAI-91, 260-267, 1991.

[11] H.Kautz and Ladkin "Integrating metric and qualitative temporal reasoning" Proceedings AAAI-91, 241-246, 1991.

[12] R.Dechter, I.Meiri and J.Pearl "Temporal constraint networks". Artificial Intelligence, 23:123-153, 1984.

A FAST SYSTOLIC ALGORITHM FOR MATRIX MULTIPLICATION

G. ISERN

UNIVERSIDAD CENTRAL DE VENEZUELA
ESCUELA DE COMPUTACION
Apartado 47002, Caracas
Venezuela

ABSTRACT

This paper describes a very fast Systolic Array for Matrix Product. Systolic Sistems usually achieve high performance by allowing computations to be pipelined over a large array of processing elements. To achieve even higher performance the systolic array proposed in this paper utilizes a column broadcasting technique plus a second level of pipelining by allowing the processing elements themselves to be pipelined. This systolic array shows not only high performance, but also optimality in terms of area and time complexity by applying recursively the fitting rows method to reduce the I/O bandwidth and speedup the matrix product algorithm

Keyword: Systolic algorithms , pipeline , matrix multiplication, speedup , VLSI.

1.- INTRODUCTION

The developments in microelectronics have revolutionized computer design. Integrated circuit technology has increased the number and complexity of components that can fit on a chip. As a result, this new technology makes it feasible to build low cost special devices to solve sophisticated problems. These devices are based on multiprocessor structures and in the use of parallel algorithms. For these reasons, our main interest is in designing parallel algorithms with simple and regular data flows and using pipelining as a general method for implementing these algorithms in hardware. The results are Systolic arrays , which is the name given by Kung [1] to a single and regular interconnection of processor elements which cooperate to carry on an especific task (matrix multiplication in this case). Pipelining a set of processor elements plus pipelining within each stage should lead to the best possible performance this is optimize area and time obtaining a great improvment with respect to previous work.

The first contribution to systolic system was made by Kung and Leiserson in 1978 [1], Suros and Montagne [2, 3] optimized the systolic networks proposed by Kung by fitting first diagonals and then fitting rows for reducing I/O bandwidth, Isern [7] and Montagne , Suros and Isern [4] obtained acceleration by the use of a systematic aligning method called row fitting which also reduce the area of the systolic array. Kung also proposed a two level pipelined systolic array [5] for an algorithm of multidimensional convolution.

Section 2 describes one level pipeline matrix-multiplication with regular structure and fitting rows, showing estimates of acceleration and Area. Section 3, describes the new proposed algorithm, the fitting rows model for a two levels piplined systolic array with broadcasting, and finally section 4 shows results and conclusions.

2.- MATRIX PRODUCT ON SYSTOLIC ALGORITHM:

Before describing systolic arrays, some concepts are to be defined:

(1) Elemental processor: a processor that executes

$$c = c + (a*b)$$

(2) $T_s(n)$: Sequencial execution time

(3) $T_p(n)$: Parallel execution time

(4) EPU: Efficiency of processor utilization

$$EPU = T_s(n)/T_p(n)*P$$

where P is the number of processors and 0 EPU 1

(5) Area: Number of processors in a systolic network

(6) IPSP : Inner product step processor.

A well-know systolic array to compute an n * n band matrix-vector multiplication AX = Y was originally reported elsewhere [8]. Figure 1 and 2 depicts this organization for a band matrix and the first seven steps of the algorithm.

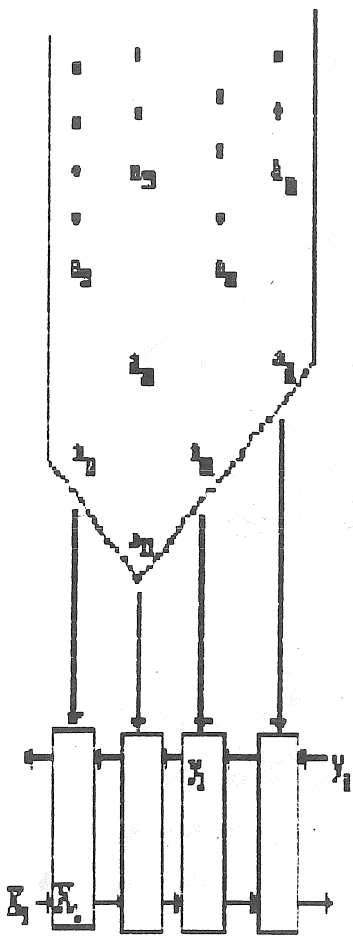


FIGURE 1

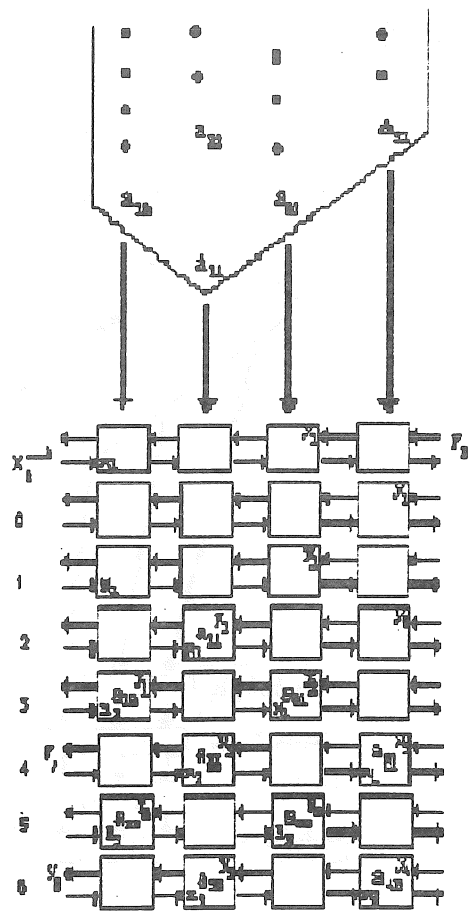


FIGURE 2

For the case of matrix multiplication it is easy to see that the matrix product $C = (c_{ij})$ of $A = (a_{ij})$ and $B = (b_{ij})$ can be computed by the following recurrences.

$$c_{ij}^{(1)} = 0,$$
$$c_{ij}^{(k+1)} = c_{ij}^{(k)} + a_{ik}\, b_{kj},$$
$$c_{ij} = c_{ij}^{(n+1)}$$

This recurrences can be evaluated by pipelining the a_{ij}, b_{ij} and c_{ij} through a systolic array having w_1. w_2 (bandwidth of A and B respectively) Hex-connected IPSP or through a systolic array having W.N (NxN matrix) orthogonally-connected IPSP (see figure 3). The latter will be the one used all along this work, this one is an extension of Leiserson/Kung matrix-vector multiplication algorithm for N vectors.

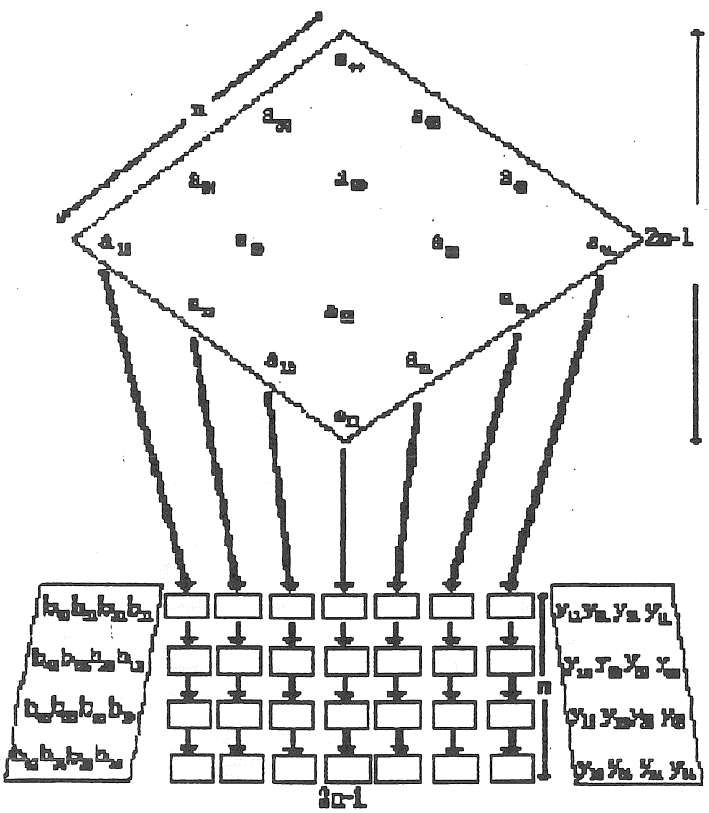


FIGURE 3

Let A = (aij) be a nxn matrix with bandwidth w = 2n-1 and X a n elements vector. It is readily seen that after w units of time (filling time FT) the results of the product Y=Ax start shifting out from the left-end processor at the rate of one output every two time units. Therefore, using our network all the n components of Y can be computed in 2n-1+w time units, which compared with the O(wn) time units needed for the sequential algorithm on a uniprocessor, is a great improvement.

Let see the formula:

a) Filling time FT = w
= 2n-1

b) Every two clock pulses (Figure 5) it is produced one result, therefore 2n cycles are necesary for the total vector
Computing Time CT = 2n - 1

c) Execution Time T(n) = FT + CT
= 4n - 2

Area = w = 2n - 1

Making an extension of the algorithm over a systolic network (adding rows), we obtain a network for multiply a matrix times several vectors, therefore an architecture for matrix product (orthogonally-connected, see figure 3).

The execution time for this model can be described in this way:

a) First Row filling time = 2n - 1

b) First vector computing time: 2n - 1

c) N-1 remaining vectors computing time: n-1

$T_m(n) = 2n - 1 + 2n - 1 + n-1 = 5n-3$

$\text{Area}_m(n) = w \times n$
$= (2n - 1)n$
$= 2n^2 - n$

Fitting Rows Method (FRM)

Proposed by Suros and Montagne (SMM, Suros/Montagne Model) in 1987, describes how modifying the processor internal architecture a_{ij} stays for two IPSP cycles eliminating the every two clocks idle processor cycle. It is required one application of FRM on B matrix to keep dataflow consistency (see Figure 4).

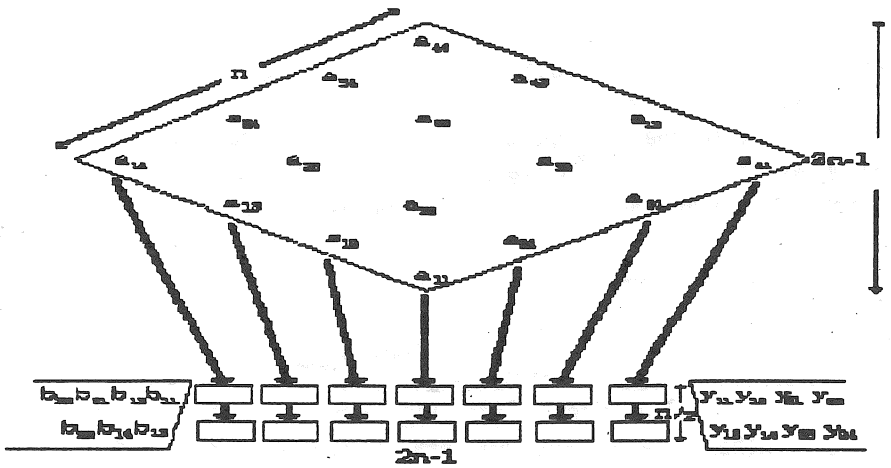


FIGURE 4

Definition

Let B = (b_{ij}) a n x n Matrix, f_1 and f_2 two adjacents n elements B rows.

$f_1 = (b_{i,1} ; b_{i,2} ; \ldots , b_{i,n})$

$f2 = (b_{i+1,1} ; b_{i+1,2} ; \ldots ; b_{i+1,n})$

A row fitting is defined as:

$f_\partial = f_1 \underline{o} f_2$

$f_\partial = (b_{i,1} ; b_{i+1} ; \ldots ; b_{i,n} ; b_{i+1,n})$

where length (f_∂) = Length (f_1) + Length (f_2)

Therefore every n rows B matrix can be transformed to n/2 rows B matrix using FRM and the execution time of the systolic network using FRM is given by:

a) First row filling time: 2n-1 (w)
b) First two vectors computing time: 2n
c) n/2 - 1 remaining vectors computing
Time : n/2 - 1

$T_{m_\partial}(n) = 2n - 1 + 2n + n/2 - 1$

$= 4.5\, n - 2$

$$\begin{aligned} \text{Area}_m^{\partial}(n) &= \text{Area}_m(n)/2 \\ &= (2n^2-n)/2 \\ &= n^2 - n/2 \end{aligned}$$

Applying recursively the FRM we reduce the I/O bandwidth to one, but with less speed up.

3. TWO LEVEL PIPELINING PLUS FITTING ROWS METHOD (TLFRM)

The two techniques to be described here will accelerate the matrix product with the same area of FRM or reducing it by applying recursively n times FRM [4].

The first model, consists in broadcasting the a_{ij} values to all IPSP in the same column of the systolic network at the same time, reducing the time for matrix multiplication to the time of matrix-vector multiplication [6] (see Figure 5)

$T_{mb}(n) = 4n-2$ (without FRM)

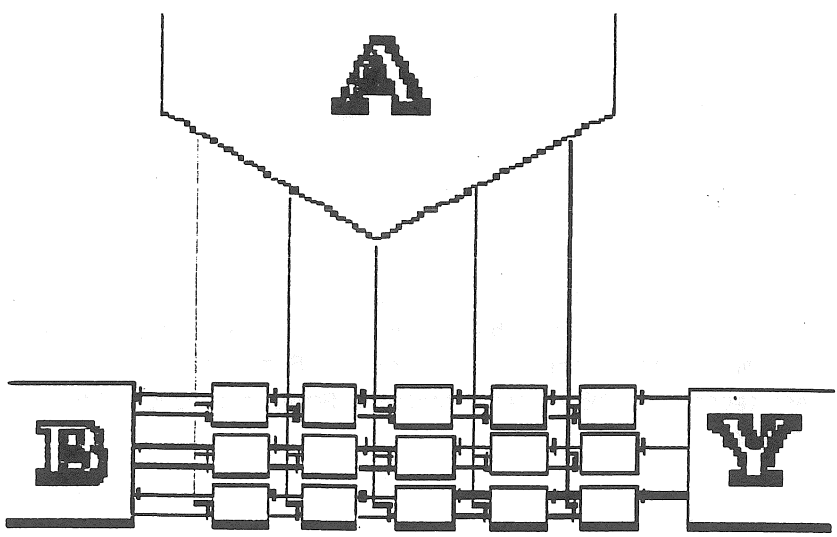


FIGURE 5

Combining the latter technique with FRM we obtain, a lot faster matrix product than with LKM (Leiserson/Kung Model) or FRM, using a smaller area, therefore better EPU is achieved.

The second method uses a second level of pipelining inside each IPSP, consisting of a two four-stages floating point operators, one for floating point addition and the other one for floating point multiplication, therefore the real pipe length is:

$$\begin{aligned} LENGTH_{2pipe}(n) &= w.8 \\ &= (2n-1).8 \\ &= 16n - 8 \quad \text{Elements} \end{aligned}$$

This change generates some minors modifications to the IPSP owing to the operands synchronization of the add operator with the multiplication result (see Figure 6). In order to synchronize it is necessary to attach a four stages shift register. The systolic network general operation will be developed using FRM.

This model executing time is:

a) Filling Time: $FT = w$
$= 2n-1$

b) First two vectors computing time is equal to the corresponding time in the FRM divided by the length of the inner pipeline (eight in this study). This is:

$$CT = 2n/8$$

c) The n/2 remaining rows computing time has two values depending on which is the chosen method, in the straight one (IMM, Isern/Montagne Model), the time is (n/2-1)/8 but combining with the broadcast technique (IMBM, Isern/Montagne Broadcast Model) this time is equal to zero.

$$\begin{aligned} T_{m\ \partial\ 2^{(n)}} &= 2n-1 + 2n/8 + n/16 - 1/8 \\ &= 2.31\,n - 1.12 \\ T_{m\ \partial\ 2b^{(n)}} &= 2n-1 + 2n/8 \\ &= 2.25n - 1 \end{aligned}$$

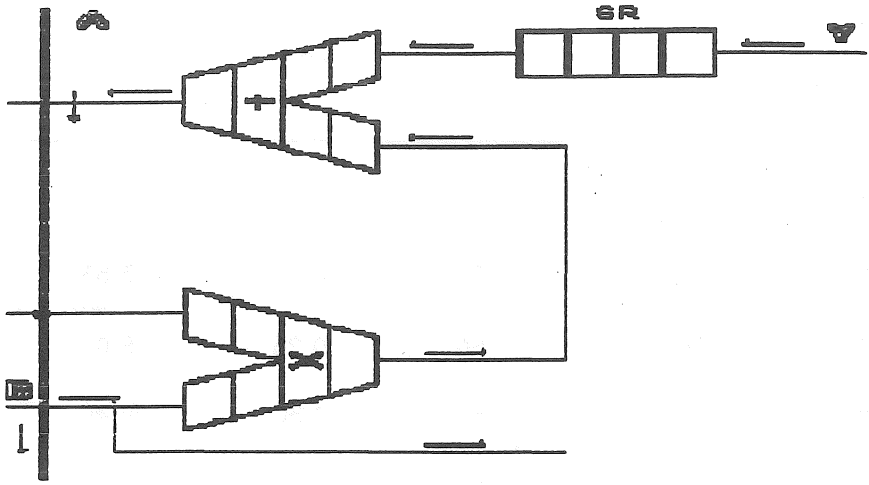


FIGURE 6

The systolic network area is equal to the one obtained by the FRM. It is possible to apply recursively the FRM reducing 50% of the precedent matrix on each application (IM2M, IM3M, Isern/Montagne Model with 2 and 3 fitting rows respectively). Combining FRM with two level pipelining method we obtain a very fast algorithm with the minimum processor area.

$$T_{m_{\partial}\ 2^{2(n)}} = 2n - 1 + 4n/8 + n/4.8 - 1/8$$
$$= 2.53\,n - 1.12$$
$$Area_{m_{\partial}\ 2^{2(n)}} = n^2/2 - n/4$$

TABLE 1 and 2 sumarizes executing times for the studied methods, in constrast with von Neumann Model, VNM.

MODEL	TIME	AREA
VNM	n^3	-
LKM	5n - 3	$2n^2 - n$
SMM	4,5n - 2	$n^2 - n/2$
IMM	2,31n - 1,12	$n^2 - n/2$
IMBM	2,25n - 1	$n^2 - n/4$
IM2M	2,53 - 1,12	$n^2/2 - n/4$

TABLE 1

MODEL	TIME	AREA	EPU	SPU
VNM	512	-	-	-
LKM	37	120	0,11	13,83
SMM	34	60	0,25	15,05
IMM	17,36	60	0,50	29,49
IMBM	17	60	0,51	30,11
IM2M	19,12	30	0,89	26,77

TABLE 2 (n=8)

Figure 7 show the executing time and figure 8 show the area.

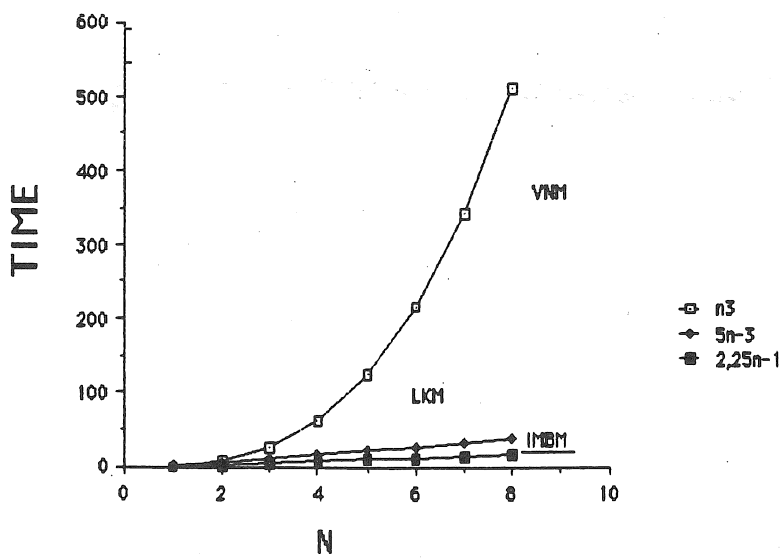


FIGURE 7

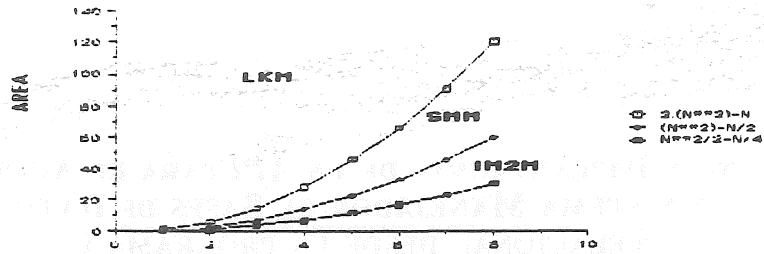


FIGURE 8

4. CONCLUSION:

In conclusion, by the application of this method to any matrix we can halve the computation time in comparison with SMM, keeping the same area, furthermore it is possible to have the minimum area with the recursively FRM application [4] and still having a better time than the SMM.

REFERENCES

[1] H.T. Kung and C.E. Leiserson, Algorithms for VLSI Processor Arrays, in: C. Mead and L. Conway, Eds., Introduction to VLSI Systems (Addison-Wesley, Reading, MA, 1980) Section 8.3 pp. 271-292.

[2] Surós, R. and Montagne, E. "Optimizing Systolic Networks by Fitting Diagonals"., Parallel Computing, North-Holland 4 (1987) 167-174.

[3] Surós, R. and Montagne, E. "Fitted Diagonals for Reducing I/O Bandwidth in Systolic Systems", Information Processing Letters, North-Holland (1987).

[4] Montagne E., Suros R., Isern G., "A deterministic model for spatial data scheduling on systolic arrays" Advances in parallel computing, volume 2 pages 291-304, 1992 JAI Press Ltd.

[5] H.T. Kung, "A two-level pipelined systolic array for multi-dimensional convolution" Department of Computer Science, Carnegie-Mellon University, USA, Nov. 1982.

[6] K.H. Cheng and S. Sahni, "VLSI Systems for band matrix multiplication", Parallel Computing, Vol. 4, Number3, June 1987 North-Holland, 239-258.

[7] Isern G. "Optimizacion del tiempo de ejecucion y del numero de procesadores en un algoritmo sistolico de multiplicacion de matrices". Tesis de Mestria, Facultad de Ciencias , UCV, 1990.

[8] Leiserson, C. 1983, Area-Efficiency VLSI Computation, ACM Doctoral dissertation award, MIT Press, Mass.

IABD :

DISEÑO E IMPLANTACION DE UN *API* PARA EL ACCESO A UN SISTEMA MANEJADOR DE BASES DE DATOS RELACIONAL DESDE UN PROGRAMA C

Claudia Lucía JIMENEZ GUARIN

Rolando BLANCO MALDONADO

Grupo de Investigación SINBAD
Departamento de Ingeniería de Sistemas y Computación
Universidad de los Andes

Apartado Aéreo 4976, Santafé de Bogotá, COLOMBIA

e-mail : cjimenez@andescol.bitnet, rblanco@andescol.bitnet

fax: (57.1).284.18.90

0. Resumen

Este artículo presenta el diseño, implantación y resultados de evaluación de un *API* (*Application Program Interface*) para el acceso a Bases de Datos Relacionales por programadores no expertos en Bases de Datos. Está desarrollado para ser utilizado desde programas C, o desde programas que puedan hacer invocación a funciones en este lenguaje. El objetivo principal es lograr en las aplicaciones portabilidad, independencia del Manejador de Bases de Datos (SMBD), facilidad y rapidez de desarrollo y compatibilidad con la sintáxis de C. Se libera además al programador del manejo complejo de variables, estructuras e instrucciones propias al ambiente de programación del SMBD, permitiendo que programadores no expertos en el manejador específico logren utilizarlo sin necesitar entrenamiento adicional. Se muestran los resultados de una implantación específica, sobre ORACLE. Actualmente se está desarrollando el *API* para otros manejadores, INFORMIX y UNIFY.

Palabras Claves: Bases de Datos Relacionales, C, Programación de Aplicaciones sobre Bases de Datos.

1. Introducción

Gran número de aplicaciones requieren para su desarrollo de las facilidades de un lenguaje de propósito general (e.gr. C, Pascal, Lisp), además de las operaciones de definición y manipulación de datos que provee un Sistema Manejador de Bases de Datos (SMBD). Cada SMBD comercial especifica su propia interface o interfaces de acceso a la Base de Datos (BD) y, aunque en su mayoría son bastante similares, existe un alto riesgo de incompatibilidad entre aplicaciones escritas para distintos SMBD. Por lo general, la utilización de estas interfaces involucra un proceso de aprendizaje adicional, un esfuerzo grande de codificación y control, así como un mayor conocimiento del SMBD sobre el que se está desarrollando la aplicación, lo que lleva a programadores no expertos a problemas adicionales que en gran medida pueden ser evitados. En este documento se presenta la especificación, diseño e implantación de una interface simple, portable y homogénea con el lenguaje C [8] [2] [7], para construir aplicaciones sobre un SMBD-Relacional [4], sin necesitar ser un experto en Bases de Datos.

Este artículo se encuentra organizado de la siguiente manera: en la segunda sección se presentan las formas existentes de acceso a un SMBD-Relacional desde un programa escrito en lenguaje C; en la tercera sección se muestra globalmente la interface propuesta, **IABD**. En la cuarta sección se especifican las funciones que son ofrecidas por IADB, la arquitectura general del API y ejemplos detallados que ilustran la utilización de IABD. Finalmente se presentan algunas consideraciones acerca de su portabilidad y las conclusiones.

2. Formas de Acceso a un SMBD-Relacional desde un Programa C

La comunicación entre un programa C y un SMBD-Relacional puede ser realizada a partir de dos técnicas [9] [11] [12] [8], excluyentes entre sí:

a. Embebiendo instrucciones SQL en el código C del programa.
b. Realizando llamados directos al SMBD en forma de funciones.

La técnica de acceso a la BD a partir del uso de SQL embebido es, por lo general, la interface de comunicación más utilizada por programas C que necesitan acceder SMBDs Relacionales. Es implantada de forma similar por la mayoría de BDs Relacionales, lo que conlleva como principal ventaja la portabilidad de los programas escritos bajo esta técnica. Sin embargo implica una etapa de

precompilación, la cual complica la organización de programas fuente, así como la depuración de la aplicación.

La técnica de acceso directo a la BD aunque no implica labor de precompilación, es por el contrario implantada de manera distinta por cada SMBD Relacional lo que hace prácticamente imposible el transporte a través de distintos SMBD de las aplicaciones que acceden directamente a la BD.

Es claro que ambas opciones involucran serios inconvenientes. El problema es aún mayor para desarrolladores no expertos en BDs. Es por lo tanto deseable una tercera opción que tenga las ventajas de las interfaces existentes y evite en lo posible los problemas que actualmente conllevan.

Estas razones nos condujeron a la especificación, diseño e implantación de una nueva forma de acceso a la BD, la interface IABD. IABD permite accesar la BDR desde un programa C con las siguientes características:

- Sin etapa de precompilación. Esta característica facilita la organización de los archivos así como la depuración de los programas desarrollados.
- Tiene una interface simple y homogénea con el lenguaje C. La especificación de una interface simple garantiza la rápida creación de código fuente para programadores familiarizados con el lenguaje C.
- El código creado debe ser de fácil mantenimiento. Cambios en los requerimientos de una aplicación desarrollada utilizando esta interface no deben generar modificaciones en los módulos encargados del acceso a la BD.
- Alivio al programador de los detalles de acceso a la BD. Característica importante, si se desea facilitar el uso de la interface a personas no familiarizadas con el manejo de BD.
- Acomodarse a las necesidades del programador. Por lo general, distintas aplicaciones necesitan interactuar de manera diferente con el SMBD. Algunas a más alto (o bajo) nivel que otras. La interface debe estar en capacidad de ofrecer alternativas de acceso a la BD que permitan una interacción al nivel deseado, según los requerimientos y experiencia del programador.
- Debe ser portable. Se debe garantizar la máxima portabilidad posible. Esto se logra independizando la interface del SMBD sobre el cual se encuentra implantada. Al expresar los requerimientos que se desean hacer sobre la BD en forma de sentencias SQL se logra algo de esta portabilidad, pues se independiza a la aplicación desarrollada del lenguaje de manipulación de datos (DML) que realmente maneja el SMBD sobre el que corre la aplicación.

Para la utilización de IABD es necesario estar familiarizado con el lenguaje C, pero sin necesidad de ser un experto. Se requieren también algunos conocimientos sobre BDs Relacionales, y en especial sobre SQL, esto con el fin de poder especificar las consultas y demás operaciones que se quieren realizar. El acceso a la BD a través de IABD requiere de los mismos conocimientos que se necesitan para acceder una BD Relacional en forma interactiva. Los usuarios de esta interface serán personas que necesiten manipular o definir datos en una BD, y que tengan requerimientos que hagan necesaria la utilización de un lenguaje de propósito general.

3. Interface Propuesta: IABD

La ***Interface de Acceso a la BD*** o ***IABD*** esencialmente define el protocolo de comunicación con la BD. Este protocolo consiste en un conjunto de funciones y estructuras de datos accesibles a varios niveles de complejidad, según lo requiera el programador.

IABD independiza al programador de los detalles de acceso a la BD al encargarse de la mayoría de aspectos relacionados con el control de la comunicación con la BD: Manejo de *buffers*, control de

códigos de error, recuperación de tuplas y conversión de tipos de datos. IABD especifica por lo tanto una recuperación automática de tuplas, la cual se realiza fundamentalmente asociando una función del programa C usuario al conjunto de tuplas resultado del requerimiento SQL, de tal forma que esta función sea ejecutada para cada tupla de dicho conjunto. Para aquellos programadores que deseen tener un control sobre la recuperación de tuplas y todos los aspectos relacionados con esta acción, IABD especifica una serie de funciones y estructuras que permiten este nivel de interacción.

4. Las Funciones

En IABD se especifican cuatro tipos de funciones:

- **Funciones Básicas :** Son utilizadas para establecer y/o terminar la comunicación con la BD, así como para especificar bajo qué ambiente se desea esta comunicación.
- **Funciones de Alto Nivel** : Son las funciones que realizan una recuperación automática de las tuplas resultado de un requerimiento SQL.
- **Funciones de Bajo Nivel** : Hacen parte de este grupo las funciones que permiten al programa C usuario el control sobre la recuperación de tuplas.
- **Funciones de Manejo de Argumentos** : IABD permite la recuperación de las tuplas resultado de un SELECT, en variables locales y globales del usuario.

Independientemente del tipo de función que se utilice, en ningún caso el manejo de la memoria involucrada en la ejecución de un requerimiento SQL queda a manos del programa usuario. En un mismo programa el usuario puede utilizar indistintamente funciones de uno o varios niveles y realizar simultáneamente tantos accesos como desee.

Finalmente el requerimiento SQL que el programa C usuario transmite a IABD, a través de funciones de alto o bajo nivel, puede estar parametrizado siguiendo una sintáxis similar a la utilizada en las funciones `printf` y `scanf` [7].

4.1. Funciones Básicas

Hacen parte de este grupo las siguientes funciones:

Funciones Básicas	
conexion	autocommit
`commit`	`impresion`

Fig. 1

La especificación de cada una de ellas es:

```
int conexion (unsigned short cod_op, char *usuario,  char *password)
```

permite establecer o terminar la conexión con la BD, dados los datos del usuario y su clave.

```
int autocommit (unsigned short cod_op)
```

activa o desactiva la opción de validación automática. Por defecto esta opción está *desactivada.*

```
int commit (unsigned short cod_op)
```

permite realizar la validación o anulación de una transacción.

```
int impresion (unsigned short cod_op)
```

activa o desactiva la opción de impresión por pantalla. Por defecto, cuando se ejecuta una operación de selección a alto nivel sobre la BD y el usuario no especifica la acción a realizar para cada tupla resultado de la selección, estas se imprimen por pantalla. Si el usuario no desea que se realice esta acción, debe informarlo a IABD por medio de esta función.

4.2. Funciones de Alto Nivel

Las funciones de Alto Nivel ofrecen la máxima independencia al programador de los detalles relacionados con el acceso a la BD. Estas funciones permiten al usuario ejecutar cualquier operación de definición, inserción, supresión, actualización o búsqueda sobre la BD, en forma de una sentencia en sintáxis SQL. Cuando la sentencia es un SELECT, la recuperación de tuplas es realizada automáticamente. Adicionalmente IABD ofrece a este nivel la posibilidad de manejar o no los códigos de error resultado de la ejecución de la sentencia SQL (Fig. 2), así como la información de control relacionada con dicha ejecución.

Funciones de Alto Nivel	
(Recuperación Automática de Tuplas)	
Sin Inf. de Control	Con Inf. de Control
`sqlo`	`sqloCntr`
`sql`	`sqlCntr`

Fig. 2

Las funciones de Alto Nivel `sql` y `sqlCntr` permiten asociar a la ejecución de una operación de selección sobre la BD una función del programa usuario para ser ejecutada con cada tupla resultado de la selección. Los argumentos de dicha función solo pueden ser conocidos por IABD al momento de ejecución, por lo que requieren de un soporte para su manejo. Este soporte es dado por las funciones de Manejo de Argumentos (ver la sección 4.3).

La interacción más simple del programa usuario con IABD es a partir de las funciones de Alto Nivel que no involucran información de control, por lo tanto las funciones `sqlo` y `sql` serán presentadas primero, para finalizar con la descripción de las funciones `sqloCntr` y `sqlCntr`, las cuales dan al programa usuario información relacionada con la ejecución de la sentencia SQL [8] [2].

```
int sqlo (char *sentencia_sql)
```

El objetivo de esta función es el de permitir, de la forma más sencilla posible, la ejecución de cualquier sentencia SQL. Por ejemplo, la creación de la tabla ESTUDIANTE con atributos código, nombre y promedio a partir de un llamado a esta función, se realiza de la siguiente manera:

```
sqlo("create table estudiante ( código    number  not null, \
                                nombre    char(30) not null, \
                                promedio  number(4,2))");
```

En caso de que la operación ejecutada sea un SELECT y la opción de impresión esté activada, las tuplas seleccionadas se imprimen por pantalla.

```
int sql (char *sentencia_sql [, parámetros] ... )
```

Esta función permite la ejecución de cualquier sentencia SQL *parametrizada*. Además, realiza una recuperación automática de tuplas en caso de que la sentencia SQL sea un SELECT. La sentencia puede contener los caracteres '&', '%', '{', indicativos de los tipos de argumentos pasados como parámetros, los cuales tienen el mismo significado que en las funciones estándar de Entrada/Salida en C (`printf/scanf`) [7].

Los tipos de parámetros permitidos son:

- Variables o constantes.
- Dirección de una función entera, la cual será ejecutada para cada tupla resultado del SELECT.
- Direcciones de variables destinadas a recibir el valor de un atributo obtenido a través de un SELECT. Estas variables pueden ser LOCALES o GLOBALES, dependiendo de si la función entera especificada por el usuario actúa sobre variables globales o locales.

Los descriptores para los parámetros son:

a. `%t` Donde t es indicativo del tipo de la variable o constante que se recibe como parámetro, se siguen las mismas convenciones que `printf`, así por ejemplo un descriptor %d indica que uno de los parámetros es de tipo entero.

b. `&nt` Donde t es indicativo del tipo de variable que recibirá el valor de un atributo obtenido a partir de un SELECT. n es un entero, necesario cuando se manejan cadenas de caracteres o de bytes, indica el tamaño del atributo a recuperar. Por ejemplo un descriptor &d indica, al igual que en la función `scanf`, una dirección de una variable de tipo entero. Las variables que se especifican deben ser globales a menos que la invocación a la función `sql` haya sido precedida por una invocación a la función `argumentos` (ver sección 4.3).

c. `{f}` Indicativo de la dirección de una función entera (una sola función es permitida como parámetro) para ejecutar con cada tupla resultado de una selección. IABD soporta la acción de esta función sobre variables tanto locales como globales del usuario. Por defecto la acción de esta función es sobre variables globales.

   El usuario mantiene el control sobre la recuperación de tuplas y puede decidir en cualquier momento interrumpirla. Al terminar la recuperación automática de tuplas se retorna el control al programa del usuario, siguiendo el flujo normal de instrucciones.

Si el usuario considera necesario tener control sobre la ejecución misma de la sentencia SQL, IADB pone a su disposición dos funciones, equivalentes a las anteriormente presentadas, que adicionalmente le ofrecen información sobre número de tuplas involucradas en la ejecución, nombre de las columnas, etc. Estas funciones son sqloCntr y sqlCntr.

```
int sqloCntr(t_estadosql *estado_sql, char *sentencia_sql)
```

Esta función corresponde a la versión de la función `sqlo` que involucra información de control, la cual se encuentra concentrada en la estructura de datos `t_estadosql` especificada por IABD [2].

La función

```
int sqlCntr(t_estadosql *estado_sql, char *sentencia_sql[,parámetros] …)
```

es la correspondiente a `sql`, que involucra información de control. Al igual que para `sql`, la sentencia SQL puede estar parametrizada. El comportamiento de esta función es idéntico al de `sql`, solo que instancia los campos de la estructura de control pasada como parámetro, con la información relacionada a la ejecución de la sentencia SQL.

4.3 Funciones Manejadoras de Argumentos

IABD propone un manejo a los argumentos de las funciones del usuario inspirado en el manejo que da C a las funciones con número variable de parámetros [7]. Sinembargo, como IABD no cuenta con una pila de ejecución en dónde guardar los argumentos que deben pasarse a la función del usuario, este debe informar a IABD sobre la existencia de dicha función y sobre los argumentos que espera recibir.

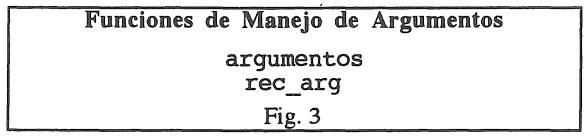


Fig. 3

Cuando el usuario desea realizar un SELECT sobre la BD, y necesita asociar a la recuperación de tuplas una función que actuará sobre variables locales, debe invocar a la función `argumentos` antes que a la función `sql` para especificar los parámetros que espera recibir su función.

La interface de la función `argumentos` es la siguiente:

```
int argumentos (char *tipos_args[, parámetros] …)
```

Donde `tipos_args` es una cadena de caracteres con descriptores del tipo **`&nt`** y/o **`%t`**, los cuales especifican los parámetros que tendrá la función del usuario.

Después de la invocación a la función `argumentos` IABD supondrá que, en el próximo llamado a la función `sql` en el que se especifique una función del usuario, esta función recibirá una pila de argumentos la cual contendrá los argumentos especificados en la invocación a `argumentos`, más *las direcciones de las variables en donde se recuperarán los atributos involucrados en el SELECT que se ejecuta*, variables que pueden ser locales.

Así por ejemplo:

```
...
argumentos("&d %f",&num_veces,prom_total);
sql("select {f} nombre &20s, código & from estudiante
     where promedio >= 4.0", imprima_estudiante,nombre_est,&código_est);
...
```

IABD supondrá que la función del usuario `imprima_estudiante` espera recibir una pila con los siguientes argumentos:

- La dirección de la variable `num_veces`.
- El contenido de la variable `prom_total`.
- La dirección de la variable `nombre_est`.
- La dirección de la variable `codigo_est`.

Los dos primeros argumentos fueron los especificados en la función `argumentos`. Los dos últimos son los que se asumen al realizar la ejecución de la función `sql`. La función `imprima_estudiante` será invocada para cada tupla resultado de la selección.

Si la invocación a la función `sql` no hubiera estado precedida por la función `argumentos`, en la ejecución de `sql` se habría supuesto que la función `imprima_estudiante` no tiene parámetros. Entonces, las variables en donde se recuperan los atributos de la selección (`nombre_est` y `código_est`) deben ser variables globales.

Para el manejo de la pila de parámetros que se pasa a la función del usuario IABD define el siguiente tipo.

```
type args_var
```

Para recibir la pila de argumentos, la función del usuario debe tener un único parámetro de tipo `args_var` (argumentos variables). Así la función `imprima_estudiante` del ejemplo, debe tener la siguiente forma:

```
int imprima_estudiante (args_var pila_args)
```

El tipo args_var puede ser definido como un apuntador a un bloque de memoria de tipo indeterminado:

```
typedef void *args_var
```

La función del usuario recupera los argumentos de la pila de argumentos a partir de la función `rec_arg`. Esta función retorna el primer argumento de una pila de argumentos de tipo `args_var`. En caso de que la pila esté vacia no se garantiza una terminación normal del programa en ejecución.

La interface de la función es la siguiente:

```
type  rec_arg(args_var pila_argumentos, type)
```

4.4. Funciones de Bajo Nivel

Las funciones de Bajo Nivel permiten al usuario administrar la recuperación de tuplas según sus necesidades. A este nivel de interacción solo es posible ejecutar selecciones (SELECTs) sobre la BD y el control de la ejecución es responsabilidad del usuario.

La ejecución y recuperación de tuplas resultado de una selección, a partir de las funciones de Bajo Nivel se realiza en dos fases:

Funciones de Bajo Nivel	
sqlb	sqls
sql_fetch	sql_abort

Fig. 4

a. Ejecución de la operación de selección, a partir de las funciones `sqlb` o `sqls`.
b. Recuperación de las tuplas resultado del SELECT (una a una), a partir de la función `sql_fetch`.

Eventualmente existe una tercera fase, en caso de que la recuperación de tuplas sea abortada intencionalmene por el programador al invocar la función `sql_abort`.

La función

```
int sqlb (t_estadosql *estadosql, char *sentencia_sql,
          tab_encabezados *encabezados)
```

ejecuta una selección sobre la BD. A diferencia de las funciones de Alto Nivel, no realiza la recuperación de las tuplas seleccionadas. La función `sqlb` es la encargada de obtener la información correspondiente a las columnas involucradas en la selección. Para cada columna se maneja el tipo y nombre del atributo [2].

La función

```
int sqls (t_estadosql *estadosql, char *sentencia_sql,
          tab_encabezados *encabezados [,parámetros] ...)
```

es muy similar a la función anterior, permite una selección SQL parametrizada con directivas de instanciación (%t). No se permiten directivas del tipo '{f}' y '&nt', ya que estas directivas se relacionan con la recuperación automática de tuplas, labor que a este nivel es realizada por el usuario. Esta función, a diferencia de la función `sqlb`, realiza una acción de instanciación. Después de esa acción su comportamiento es idéntico al de `sqlb`.

La función

```
int sql_fetch (t_estadosql estado_sql, int identificador,
                tab_buffers *buffers)
```

es la encargada de la recuperación de cada tupla; para tal efecto debe manejar una estructura que contenga los buffers en donde se almacenan los valores de los atributos de la tupla recuperada [2].

La función recupera la siguiente tupla resultado de la selección. En caso de que no queden tuplas por recuperar, o se presente algún error, retorna un descriptor inválido (negativo), y actualiza el campo `error` en `estado_sql`.

La función

```
void sql_abort (int identificador)
```

permite abortar la recuperación de tuplas y se encarga de liberar la memoria reservada para la recuperación de tuplas. El parámetro `identificador` corresponde al identificador de la sentencia SQL que se desea abortar.

5. Implantación

IADB fue implantado sobre ambiente Unix Versión 5 de AT&T [1], para el SMBD Relacional ORACLE Versión 6.0, accesado a partir de la librería OCI versión 1.1 [9]. El compilador utilizado fué gcc versión 1.40 [6]. Compilaciones sobre DOS se realizaron con turboC [3]. Las medidas de rendimiento nos indican que la carga adicional a la aplicación, con la utilización de IABD, no supera el 8% en el peor de los casos, siendo un 1% la sobrecarga normal.

IABD fue implantado siguiendo una arquitectura por capas, de tal manera que las funciones de bajo nivel implantan las funciones de alto nivel. La comunicación entre los módulos es netamente funcional, de tal manera que es explícita la forma de interacción entre los distintos componentes. Desde el punto de vista del desarrollo mismo del API, y en aras a lograr un máximo de portabilidad, los puntos de dependencia de un SMBD específico son minimizados.

Los grados de dependencia los podemos clasificar de la siguiente manera:

a - Con respecto a la Máquina y el Sistema Operacional.

Para lograr independencia de la máquina y del sistema se realizó la implantación en Ansi-C. Así, una aplicación desarrollada sobre ORACLE en ambiente UNIX, será compilable y ejecutable sobre el mismo SMBD en ambiente DOS o VMS. No se recomienda implantar IABD en lenguajes distintos a Ansi-C, ya que se pierde la portabilidad que garantiza este ambiente de desarrollo.

b- Con respecto al SMBD.

Cada implantación de la Interface se supone totalmente dependiente del SMBD sobre la que se implantó. Así por ejemplo una implantación sobre UNIFY [5] no podrá utilizarse cuando se desea acceder a una BD SQL/DS [12]. Sin embargo, a nivel de aplicaciones, implantaciones sobre distintos SMBD serán totalmente compatibles. Esto implica que una aplicación desarrollada sobre una implantación de la interface en un SMBD específico, puede ser utilizada

sobre cualquier otro SMBD sobre el que exista una implantación de IABD, siendo necesario solamente recompilar y reencadenar la aplicación.

6. Conclusiones

La interface IABD es la implantación de una forma de acceso directo a un SMBD Relacional. Esta interface no requiere de etapa de precompilación, es homogénea con el lenguaje C, es portable y permite una interacción a varios niveles con la BD. Existe una implantación sobre ORACLE en ambiente UNIX.

Una de las mayores bondades de IABD es el encapsular completamente los detalles de acceso al SMBD, de tal manera que se garantiza uniformidad en el desarrollo de software e independencia con respecto a un Manejador en particular, con un costo prácticamente despreciable en el rendimiento de la aplicación. Además, se logra un fácil mantenimiento y desarrollo al evitar al programador el aprendizaje de los detalles de conexión e interacción con el SMBD.

IABD está actualmente en etapa de implantación sobre UNIFY e INFORMIX y es utilizado en el desarrollo de varias aplicaciones usuarias de SMBD, entre ellas aplicaciones transaccionales distribuídas [10].

Referencias

[1] M. J. Bach. "The design of the Unix Operating System". Ed. Prentice Hall, 1986.

[2] R. Blanco. "Interface de Acceso a un Sistema Manejador de Bases de Datos Relacional desde un Programa C". Proyecto de Grado ISC-91-II-03 Ingeniero de Sistemas y Computación Universidad de los Andes, febrero de 1992.

[3] "Turbo C Reference Guide, Version 2.0". Borland International, 1988.

[4] E.F. Codd. "A Relational model for Large Shared Data Banks". ACM Vol. 13, Num. 6 (jun.1970).

[5] UNIFY. "Direct HLI Reference".

[6] GNU, Compilador gcc, obtenido vía FTP-anónimo.

[7] B. Kernighan, D. M. Ritchie. "The C Programming Language" (Second Edition). Prentice-Hall.

[8] M. C. Pajon. "OPIDUM Une Interface Procedurale de Haut Niveau pour ORACLE dans un Environnement Distribué". Centre de Recherche Groupe, Grenoble Francia, 1988.

[9] Oracle Corporation. "ORACLE References". 1988.

[10] C. Franky, J. Abásolo, R. Cucalón, G. Acosta, S. Maya. "PERSEO: Ambiente de Programación de Aplicaciones Transaccionales Distribuídas sobre Bases de Datos Relacionales". XVIII Conferencia Latinoamericana de Informática, Las Palmas de Gran Canaria, 1992.

[11] Digital Corp. "Relational Data Manipulation Language".

[12] IBM Corp. "SQL/Data System Application Programming for VM/System Product".

Conference:	XVIII Latinamerican Informatics Conference
Date:	August 31 to September 4, 1992
Location:	Las Palmas de Gran Canaria
Main Topic:	Databases
Keywords:	Object Oriented Database, CASE-Tool, Datamodelling

BIER: The Executable Conceptual Data Model Based on an Object Oriented Database System

Friedrich Kastner
Paul Andlinger
Christian Gierlinger
Werner Retschitzegger

FAW
Forschungsinstitut für
Anwendungsorientierte Wissensverarbeitung
Johannes Kepler Universität Linz
Altenbergerstraße 69
A-4040 Linz Austria

Abstract:

During the last years database systems which have previously been limited to comercial and administrative use have made their way into new application domains. On one hand existing data models (especially the relational model) have to be extended and on the other hand new database tools have to be investigated for coping with new

requirements. Recently special attention has been paid to object oriented data models and to the implementation of powerful CASE tools for relational database systems.

In this paper we describe the possibility of combining both concepts and adding dynamic aspects to database systems. This approach results in an executable conceptual data model ("BIER-CASE-Tool"), which represents a dynamic object administration where the routine work (e.g. update processes) is automatically done by the system. In both phases, data modelling and data manipulation, the user works with a special graphic interface.

Our BIER-CASE-Tool works with an underlying object oriented database. The basis of this tool is given by the dynamic data model BIER (Behaviour Integrated Entity Relationship Model) which is able to represent static as well as dynamic aspects of real world objects. The dynamic part which handles the behaviour of objects is described by a petri-net like diagram, called the B-diagram. During the phase of data modelling the user builds this diagram by working with our system.

A further new investigation is that the B-diagram is not only used for drawing and generating the data model, but also for data manipulation. That means, it is not necessary to know any query language because the user is able to do all the database manipulation work by using the graphic interface (the B-diagram).

Figure 1 gives an overview about the components of our system. The kernel of the system is the BIER-library, which holds all functions that are destinated to executing the actions the user performs by generating and working with the B-diagram at the database level. In the system tables mainly the information about the B-diagram is stored.

The underlying object oriented database is DAMOKLES. Programming was done in a C environment. The hardware platform used is a SUN SPARC - workstation.

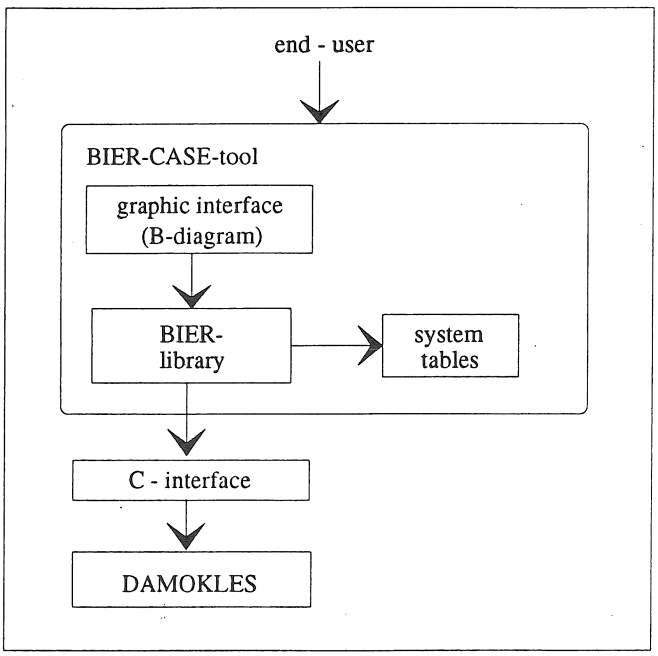


figure 1

The full version of the paper gives an overview about the BIER data model, describes the graphic interface and how the problem of the internal representation of data is solved. The last chapter gives information about results of this project and further activities in this direction.

References

/EDER 87/ EDER J., KAPPEL G., TJOA A M., WAGNER R.R.: A Behaviour Design Methodology for Information Systems, Proc. IEEE 6th, Int. Conference on Computers and Communications, Phoenix, 1987.

/STONEBRAKER 90/ Stonebraker M., Rowe L.A., Lindsay B., Gray J., Carey M., Brodie M., Bernstein P., Beech D.: Third-Generation Database System Manifesto. SIGMOD RECORD, Vol 19, No. 3, September 1990.

/ABRAMOWICZ 88/ Abramowicz K., Dittrich K.R., Gotthard W., Härtig M., Längle R., Ranft M., Raupp T., Rehm S.: DAMOKLES - Database Management System for Design Applications, Reference Manual, FZI Forschungszentrum Informatik an der Universität Karlsruhe, March 1988.

Algoritmos Pararelos Adaptativos de Ordenamiento

David A. Koufaty †

Resumen

En este trabajo se presentan algoritmos paralelos para el problema de ordenación, basados en algoritmos secuenciales adaptativos. Algunos de los algoritmos fueron implementados sobre una red de transputadores y se compara su rendimiento con otros algoritmos de ordenación secuencial.

Keywords: algoritmos paralelos, algoritmos de ordenamiento, algoritmos adaptativos de ordenamiento

1 Introducción

Se dice que un algoritmo de ordenamiento es adaptativo si ordena una secuencia en tiempo proporcional a la longitud y el desorden de la misma; es decir, si la entrada está relativamente ordenada el algoritmo debe ser considerablemente más rápido que cuando no lo está, llegando en el peor caso a orden $O(n \log n)$ para una secuencia de longitud n. Este tipo de algoritmo es de interés intrínseco ya que la identificación de casos "fáciles" llevaría a algoritmos muy eficientes.

Para medir el desorden de una secuencia dada se han propuesto numerosas métricas que definen el preordenamiento de una secuencia [Man85, Ski88], entre ellas el número de pares de elementos en orden incorrecto o inversiones (*Inv*), el número de subsecuencias contiguas ascendentes o corridas (*Runs*), el mínimo número de eliminaciones de elementos para dejar una subsecuencia ordenada (*Rem*), la máxima distancia de un elemento a su posición correcta en el ordenamiento (*Max*), el mínimo número de intercambios para ordenar la secuencia

†Departamento de Computación, Universidad Simón Bolívar, Apartado Postal 89000, Caracas 1080A, VENEZUELA

(*Exc*), el mínimo número de listas anidadas (*Enc*) y el mínimo número de subsecuencias monótonas (ascendentes o descendentes) requeridas para cubrir la secuencia (*SMS*).

En base a la definición formal de una métrica, en [Man85] se presenta el concepto de optimalidad de un algoritmo de ordenamiento con respecto a una métrica de preordenamiento (por ejemplo, ordenamiento por inserción local es óptimo con respecto a *Inv*, *Runs* y *Rem*). Por otra parte se ha demostrado que algunas métricas son algorítmicamente más finas, en el siguiente sentido: si M_1 es más fina que M_2, entonces todos los algoritmos optimales con respecto a M_1 lo son con respecto a M_2 (por ejemplo, un algoritmo optimal con respecto a *SMS* es optimal con respecto a *Runs* y *Max* y puede ser fácilmente modificado para ser optimal con respecto a *Rem* y *Exc*), por lo cual es más interesante encontrar algoritmos optimales con respecto a métricas muy finas.

En la literatura puede encontrarse varios algoritmos secuenciales adaptativos para ordenamiento. El objetivo de este trabajo consiste en presentar versiones paralelas de algunos de ellos, en particular: *Melsort* [Ski88], *Slabsort* [LP90] y *Randomized Slabsort* [ECW91]. Consideraremos además que el orden en el cual deben quedar los elementos de la secuencia es no–decreciente.

2 Partición de S

Los algoritmos paralelos que siguen requieren la solución paralela del siguiente problema: dada una secuencia cualquiera S de tamaño n distribuída uniformente entre p procesadores ($S = S_1 S_2 \ldots S_p$ y $|S_i| = n/p$) y dados $p-1$ pivotes $t_1, t_2, \ldots, t_{p-1}$, tales que $t_1 \leq t_2 \leq \ldots \leq t_{p-1}$ redistribuir la secuencia S en subsecuencias $S'_1, S'_2, \ldots, S'_p$, asignadas a los p procesadores, de manera que todos los elementos resultantes en el procesador i, s'_{ij}, $j = 1, \ldots, |S'_i|$, sean tales que $s'_{1j} \leq t_1$, $t_{i-1} \leq s'_{ij} \leq t_i$ si $2 \leq i \leq p-1$ y $t_{p-1} \leq s'_{pj}$ (es decir, los elementos estan particionados según los pivotes y todos los elementos de una subsecuencia son menores o iguales a los de la siguiente). Observe que algunas de estas subsecuencias pueden ser vacías.

En primer lugar los procesadores deben determinar donde deben quedar distribuídos sus elementos. Para ello se requiere una difusión (*broadcast*) de los pivotes, y luego una búsqueda del procesador correcto para cada elemento, lo cual puede hacer independientemente

cada procesador en $O(|S_i| \log p) = O(n/p \log p)$ comparaciones usando búsqueda binaria sobre los pivotes. Por último es necesario un intercambio total (*total exchange*) entre los procesadores para redistribuir los elementos nuevos, el cual requiere $p(p-1)$ comunicaciones unidireccionales y $2(p-1)$ pasos de comunicación sobre una topología de conexión completa (ya que sólo se pueden hacer $p/2$ comunicaciones síncronas en paralelo). Si denotamos con *local_distribution*(S_i) a la partición de la secuencia S_i en un procesador y con *send(x,y)* al envío de los elementos del procesador x que deben ir al y, entonces la partición *partition*(S,t_1, ..., t_{p-1}) se obtiene con:

broadcast(t_1, ..., t_{p-1})
for $i = 1$ **to** p **do in parallel**
    local_distribution(S_i)
endfor
for $j = 1$ **to** $p - 1$ **do**
    for $i = 1$ **to** p **do in parallel**
        if $(i/j) \bmod 2 = 0$ **then** *send(i,i + j)*
    endfor
    for $i = 1$ **to** p **do in parallel**
        if $(i/j) \bmod 2 = 1$ **then** *send(i,i + j)*
    endfor
endfor

Si el número de pivotes es mayor o igual al p, el algoritmo puede ser modificado fácilmente asignando uniformemente los pivotes a los procesadores.

Una característica del algoritmo es su falta de estabilidad (la posición relativa de los elementos en S'_i no es necesariamente la misma que en S), debido a que el orden final de los elementos en S'_i será el orden de recepción de los mismos durante el intercambio total (si exigimos que sea estable es necesario serializar gran parte de la comunicación).

3 *Melsort*

Denotaremos como *melsort(S,n)* al algoritmo secuencial *Melsort* sobre una secuencia S de n elementos y como *melpar(S,n,p)* a la versión paralela propuesta en este trabajo sobre una secuencia S de n elementos usando p procesadores.

Una secuencia de listas anidadas es un conjunto ordenado de listas ordenadas $l_1, \ldots, l_m$ tales que l_{i+1} esta anidada en l_i, es decir, el primer elemento de l_i es menor que el primero de l_{i+1} y el último elemento de l_i es mayor que el último de l_{i+1}, para todo $i = 1, \ldots, m-1$.

Melsort construye un secuencia de listas anidadas y luego procede a su mezcla (*merge*). Durante la construcción de las listas, suponiendo que se han insertado los primeros $j-1$ elementos, el elemento j-ésimo es agregado buscando la lista más vieja en la cual pueda ser agregado al principio o al final, preservando el ordenamiento de la lista (para ello se utiliza búsqueda binaria sobre los extremos de las listas); si no puede ser anexado en ninguna de las listas, se crea una nueva lista que lo contenga. El orden del algoritmo es $O(n \log m)$, donde m es la métrica *Enc* aplicada a la secuencia S; este orden esta acotado por $O(n \log n)$ y es adaptativo. Sin embargo la optimalidad del algoritmo con respecto a *Enc* es todavía una pregunta abierta.

La primera versión paralela particiona la secuencia de entrada en p subsecuencias aleatorias $S_1, \ldots, S_p$ de igual tamaño n/p y las distribuye entre los procesadores; cada procesador aplica localmente *melsort(S_i,n/p)*. Al finalizar esta fase se aplica un algoritmo paralelo de mezcla optimal [AS87].

for $i = 1$ **to** p **do in parallel**
    melsort(S_i,n/p)
endfor
parallel_merge(S)

La segunda versión particiona la secuencia de entrada S en p subsecuencias $S'_1, \ldots, S'_p$ de tamaño variable, usando para ello $p-1$ elementos aleatorios, $a_1, \ldots, a_{p-1}$, escogidos como pivotes; sólo resta aplicar localmente *Melsort*.

partition($S, a_1, \ldots, a_{p-1}$)

for $i = 1$ **to** p **do in parallel**

melsort(S'_i,n/p)

endfor

4 *Slabsort*

Slabsort particiona establemente la secuencia original S en $r = \lceil k^2/s \rceil$ subsecuencias $S_1, \ldots, S_r$ de casi el mismo tamaño, donde k es una cota superior para *SMS* de S. Para hacer esto se usan los elementos $(n/r + 1)$-ésimo, $(2n/r + 1)$-ésimo, ..., $((r - 1)n/r + 1)$-ésimo como pivotes, lo cual se hace a su vez hallando medianas repetidamente. Para cada intervalo se aplica *melsort*(S_i,$|S_i|$) con una variante que trata de ordenar la secuencia usando a lo sumo k listas anidadas; si no lo logra la variante retorna un valor que así lo indica. La selección de r, asegura que al menos la mitad de estas llamadas son exitosas; en aquellas partes donde *Melsort* no triunfa se aplica recursivamente *Slabsort*. Como inicialmente no se conoce k, el algoritmo adivina exponencialmente su valor, es decir, iniciando con un valor de $k = 2$, si en una llamada a *Slabsort* menos de la mitad de las llamadas a *Melsort* son exitosas, entonces *Slabsort* es abortado, el valor de k duplicado y el algoritmo repetido. El costo de este algoritmo es $O(n \log k)$ comparaciones, lo cual es óptimo con respecto a *SMS*.

La versión paralela se divide en dos pasos: en primer lugar hallar las medianas y particionar en subsecuencias; en segundo lugar se procede secuencialmente a aplicar una versión de *melpar*(S_i,$|S_i|$,p) con una cota de k para m en cada sección resultante; en alquellos casos donde no sea exitoso se aplica el procedimiento recursivamente. Para el primer paso se puede usar uno de los algoritmos de selección paralela o la adaptación de uno de ellos para el caso de la mediana [AS87, Akl89].

Alternativamente el último paso puede ser paralelizar el ordenamiento de cada S_i utilizando *melsort*(S_i,$|S_i|$); dado que $r \neq p$, cada procesador debe ir seleccionando secuencialmente la subsecuencia a ordenar, por lo cual debe existir un administrador de la asignación (en el caso de ser dinámica) o ser prefijada cuando comienza el algoritmo (por ejemplo, de S_1 a $S_{r/p}$ para el primer procesador, de $S_{r/p+1}$ a $S_{2r/p}$ para el segundo, etc).

En cualquiera de los casos k es también adivinado exponencialmente.

5 *Randomized Slabsort*

Randomized Slabsort es similar a *Slabsort* excepto que escoje los pivotes aleatoriamente, particionando establemente en base a ellos. A pesar de que los intervalos son ahora de longitud variable, en el caso promedio el orden del algoritmo sigue siendo $O(n \log k)$, lo cual se conjetura que es óptimo con respecto a *SMS* en el caso promedio.

La versión paralela, similar a la anterior, sólo requiere la particion en subsecuencias para el primer paso. Si suponemos que los datos están inicialmente distribuídos entre todos los procesadores, cada uno de ellos escoje independientemente r/p pivotes aleatorios, procediendo luego a la partición.

6 Resultados

El algoritmo paralelo *melpar* fue implantado somo una plataforma de tres transputadores T800 conectados en *pipeline*, cada uno con un *megabyte* de memoria. La versión implantada corresponde a la segunda, donde se aplica el algoritmo de partición señalado usando, en este caso, el número de procesadores menos un pivotes.

Se escogieron varios conjuntos de datos para probar los algoritmos para mostrar las bondades y desventajas del algoritmo: aleatorios, ordenados, invertidos, etc. Por razones de espacio sólo incluímos aquí los resultados obtenidos en el caso aleatorio y en el caso donde la secuencia está semiordenada, constando de tres corridas decrecientes de igual longitud, los cuales se muestran en la figura 1.

En ellas se muestran los tiempos de ejecución de *quicksort* y *melsort* en un tranputador y de *melpar* en tres transputadores, dando el tiempo de ejecución en pulsos de reloj, los cuales son aproximadamente 15000 por segundo. Los algoritmos fueron probados con datos relativamente pequeños debido a las limitaciones de memoria para los algoritmos secuenciales, partiendo con 30000 mil elementos y continuando con incremento iguales hasta 210000.

Como se desprende de los resultados, la eficiencia del algortimo no fue la esperada. Sin embargo todavía falta depurar la implementación y una de las razones por las cuales

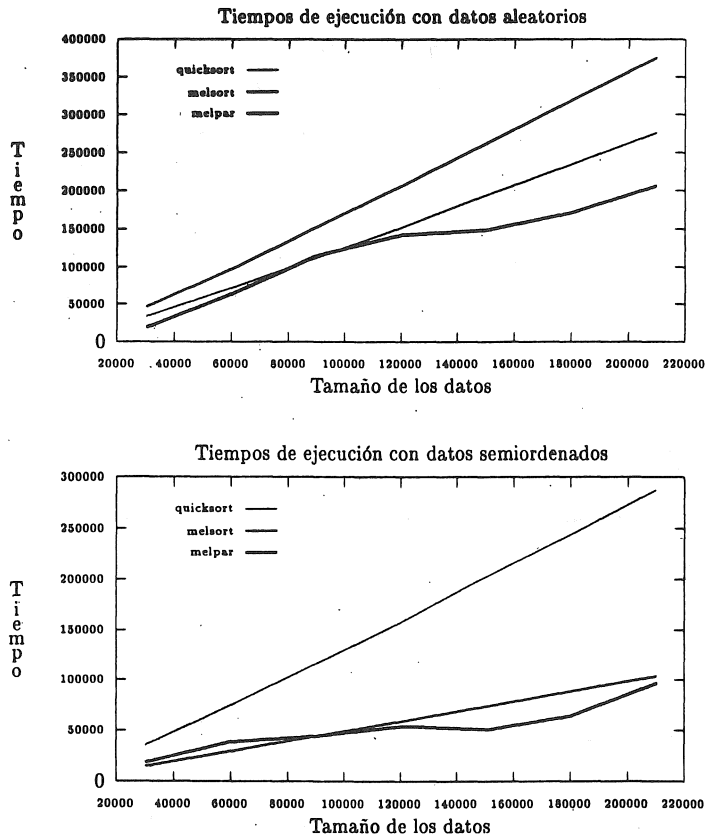


Figura 1: Resultados obtenidos

los resultados no son buenos es la topología de interconexión, la cual convierte a uno de los transputadores en un cuello de botella. La poca diferencia entre el algoritmo paralelo y el secuencial *melsort* puede deberse a la depuración que fue objeto el secuencial, logrando disminuir en 20% el tiempo de ejecución del mismo, mientras que con el paralelo sólo se tiene una versión preliminar.

Otro resultado interesante fue la necesidad de seleccionar los pivotes de manera de minimizar la comunicación, por lo cual se requiere de la implantación de un algoritmo paralelo para búsqueda de medianas.

Referencias

[Akl89] Selim Akl. *The Design and Analysis of Parallel Algorithms.* Prentice–Hall, Inc., Englewood Cliffs, New Jersey, 1989.

[AS87] Selim G. Akl and Nicola Santoro. Optimal parallel merging and sorting without memory conflicts. *IEEE Transactions on Computers*, C–36(11):1367–1369, 1987.

[ECW91] Vladimir Estivill-Castro and Derick Wood. Randomized sorting of shuffled monotone sequences. In *Proceedings of the XI International Conference of the Chilean Computer Science Society*, pages 147–155, 1991.

[LP90] C. Levcopoulos and O. Petersson. Sorting shuffled monotone sequences. In J.R. Gilbert and R. Karlsson, editors, *Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory*, pages 181–191. Springer–Verlag Lecture Notes in Computer Science 447, 1990.

[Man85] Heikki Mannila. Measures of presortedness and optimal sorting algorithms. *IEEE Transactions on Computers*, C–34(4):318–325, 1985.

[Ski88] Steven S. Skiena. Encroaching lists as a measure of presortedness. *BIT*, 28:775–784, 1988.

Animação da Dinâmica do Movimento e Colisões de Corpos Rígidos

Robson R. Lemos *
Rui Manoel R. de Bastos †
Carla M. D. S. Freitas ‡
Flávio R. Wagner §

RESUMO

Os métodos de controle de movimento em animação baseados na física, e utilizados em computação gráfica, têm como objetivo simular o comportamento de objetos de acordo com as leis físicas que governam o mundo virtual adotado. Este trabalho utiliza a dinâmica de corpos rígidos como método de controle de animação por computador aplicada a movimentos de corpos rígidos (não articulados). São apresentadas estratégias que estendem este método para possibilitar o tratamento de interações (colisões) entre corpos rígidos ou dos mesmos com o ambiente.

PALAVRAS-CHAVE: Animação, Simulação, Dinâmica, Corpos rígidos

*Bacharel em Ciência da Computação, UFSC, BR, 1987 e atualmente cumprindo o programa de mestrado na UFRGS, BR.

†Mestre em Ciência da Computação, UFRGS, BR, 1992.

‡Mestre em Ciência da Computação, UFRGS, BR, 1982, atualmente cumprindo o programa de doutorado na UFRGS, BR, e professora do Curso de Pós-Graduação em Ciência da Computação, UFRGS, BR.

§Doutor em Ciência da Computação, Kaiserlautern, RFA, 1983 e professor do Curso de Pós-Graduação em Ciência da Computação, UFRGS, BR. Instituto de Informática - UFRGS, Curso de Pós-Graduação em Ciência da Computação - CPGCC, Av. Bento Gonçalves, 9500 Bloco IV - Agronomia - Campus do Vale, CEP 91501 - Porto Alegre - RS - Brasil, Caixa Postal: 15064 FAX: (051) 336-5576, E-mail:robson@inf.ufrgs.br

1 INTRODUÇÃO

Quando um animador descreve o movimento de objetos no tempo e no espaço sem levar em conta as forças e torques que causam este movimento, ele é obrigado a usar de sua intuição para produzir movimentos próximos da realidade. No momento que surgem movimentos mais complexos a serem descritos, esta descrição explícita do movimento pode ser substituída por outros métodos de controle de movimento [2].

Este trabalho apresenta as características necessárias para a modelagem com simulação física de um sistema dinâmico de corpos rígidos sob a ação de forças e torques.

A abordagem dada ao problema da simulação do comportamento fundamental de corpos rígidos num dado mundo virtual permite a incorporação de outras características aos objetos: reações realísticas durante possíveis interações e articulações para produção de movimentos de corpos rígidos articulados.

2 SIMULAÇÃO DINÂMICA DE CORPOS RÍGIDOS

Para que a simulação dinâmica ocorra, devem estar associadas aos objetos (corpos rígidos), as seguintes características: atributos geométricos (posições e orientações), atributos físicos (propriedades de massa), estímulos (forças e torques) e restrições (vínculos) que atuam no mundo virtual (ambiente). Estas características determinam a descrição do estado do sistema dinâmico em algum instante de tempo. A partir deste estado dinâmico, é possível determinar sua evolução ao longo do tempo através de equações dinâmicas de movimento.

2.1 MODELAGEM COM SIMULAÇÃO FÍSICA

Como é abordada a modelagem com simulação física para corpos rígidos, o sistema físico deve estar condizente com a dinâmica clássica de corpos rígidos [8] e [4]. Para a modelagem são realizadas as seguintes etapas:

- Descrição física do objeto: um corpo rígido pode ser visto como uma coleção de partículas discretas ou como uma distribuição contínua da matéria. Desta forma, sua massa total não é concentrada em um único ponto, mas distribuída no espaço, tornando, assim, seu movimento rotacional significante. No espaço tridimensional um corpo rígido apresenta seis graus de liberdade (coordenadas independentes). A posição do corpo pode transladar, no sistema de coordenadas do universo (SRU), em três direções (X, Y e Z) e a orientação do corpo pode variar em três ângulos independentes, os quais fornecem a orientação do sistema de coordenadas do universo (SRU) em relação ao sistema de coordenadas do corpo (SRO). São considerados objetos com geometria simples modelados geometricamente por uma malha de pontos conectados para formar polígonos e com distribuição de massa homogênea. As propriedades de massa necessárias para a simulação são as seguintes: massa total, centro de massa

e tensor de inércia. O tensor de inércia é responsável pela distribuição de massa do corpo rígido. Uma vez determinado o tensor de inércia, no sistema de coordenadas do corpo, ele não se modifica durante o movimento.

- Formulação das equações dinâmicas : para formular as equações dinâmicas é utilizada a formulação newtoniana. Esta formulação é mais apropriada para movimentos simples e por isto é, normalmente, utilizada na simulação de partículas [5] e corpos rígidos [1], [6] e [5].

- Resolução das equações dinâmicas: as estratégias de resolução das equações dinâmicas são definidas segundo critérios relacionados aos métodos de controle de movimento (vide seção 2.2) utilizados na simulação dinâmica. Atributos físicos, tais como, massa total, centro de massa e tensor de inércia são considerados variáveis conhecidas das equações de movimento e podem ser assumidos constantes. As variáveis desconhecidas das equações de movimento são definidas de acordo com as estratégias adotadas pelos métodos de controle de movimento aplicados à simulação dinâmica.

- Integração das equações dinâmicas: para produzir o movimento dos objetos, são resolvidos sistemas de equações diferenciais de primeira ordem (vide seção 3), acopladas ou não, utilizando-se o método de Runge-Kutta de quarta ordem [11].

2.2 MÉTODOS AUTOMÁTICOS PARA CONTROLE DE MOVIMENTOS

Durante o movimento de corpos rígidos podem surgir interações (tais como, colisões) destes objetos com outros objetos ou com o ambiente. Para realizar um controle automático destas interações utiliza-se métodos de controle de movimento que permitam a combinação de respostas físicas automáticas (por exemplo, objeto caindo com a força da gravidade, forças e torques resultantes do impacto entre objetos) com sugestões de controle especificadas pelo usuário (por exemplo, forças e torques externas atuando no movimento de um objeto). Conforme citado por Wilhelms [12], estes métodos de controle operam em baixo nível (em termos de controle de movimentos dos graus de liberdade de um único corpo rígido ou de corpos rígidos articulados com poucas articulações) e em alto nível (em termos de controle da locomoção de corpos rígidos articulados ou de controle comportamental). Para este trabalho são utilizados métodos de controle de movimento que operam em baixo nível, tais como, análise dinâmica e detecção de colisões e resposta.

3 DINÂMICA DO MOVIMENTO DE CORPOS RÍGIDOS

Como vimos acima, um corpo rígido apresenta seis graus de liberdade. Para representar as novas posições e orientações dos objetos, ao longo do tempo, são utilizados dois sistemas de coordenadas: sistema de coordenadas do universo (denominação do sistema de coordenadas inerciais, fixo no espaço, para computação gráfica) e o sistema de coordenadas do corpo (fixo no corpo, com sua origem no centro de massa do corpo). Desta forma, as coordenadas

do corpo podem ser transformadas para coordenadas do universo através de rotações e translações:

$$\vec{P} = \vec{c_i} + R_i \vec{P'}$$

onde para um corpo i, $\vec{P}$ é o ponto do objeto representado no SRU, $\vec{c_i}$ é o vetor do centro de massa do corpo posição representado no SRU, R_i é a matriz de rotação (orientação) e $\vec{P'}$ é um ponto do objeto representado no SRO.

Esta divisão, em translações e rotações para representação geométrica de um objeto em computação gráfica pode ser estendida para resolução da dinâmica do movimento.

O movimento translacional de um corpo rígido é determinado por forças externas atuando em um ponto de massa do corpo localizado em seu centro de massa. Assim sendo, a dinâmica do movimento translacional de corpos rígidos sob ação de forças externas pode ser determinada resolvendo-se um conjunto de equações diferenciais ordinárias acopladas:

$$m\frac{d\vec{v_i}}{dt} = \sum_j F_j$$

$$\frac{d\vec{c_i}}{dt} = \vec{v_i}$$

onde para um corpo i, F_j são os vetores de forças externas atuando no corpo, m é a massa total do corpo, $\vec{v_i}$ é o vetor velocidade linear do corpo e $\vec{c_i}$ é o vetor de centro de massa do corpo (posição em função do tempo). Todos estes vetores são representados no SRU.

O movimento rotacional sobre o centro de massa de um corpo rígido é determinado por torques externos atuando no corpo. Os torques externos, $\vec{T_i}$, resultam de forças externas do corpo, $\vec{F_i}$, atuando em um ponto de massa do corpo, $\vec{P_i}$ (em uma localização diferente do centro de massa):

$$\vec{T_i} = \vec{p_i} \times \vec{F_i}$$

Dessa forma, a dinâmica do movimento rotacional de corpos rígidos sob ação de torques externos pode ser resolvida no sistema de coordenadas do corpo (SRO), através de um conjunto de equações diferenciais ordinárias acopladas conhecidas como equações de Euler [8]:

$$\begin{aligned} I_x\frac{d\vec{\omega}_x}{dt} + (I_z - I_y)\vec{\omega}_y\vec{\omega}_z &= \vec{T}_x \\ I_y\frac{d\vec{\omega}_y}{dt} + (I_x - I_z)\vec{\omega}_x\vec{\omega}_z &= \vec{T}_y \\ I_z\frac{d\vec{\omega}_z}{dt} + (I_y - I_x)\vec{\omega}_x\vec{\omega}_y &= \vec{T}_z \end{aligned}$$

onde para um corpo i, $\vec{T}$ são os vetores de torques externos atuando no corpo, I são os momentos de inércia sobre o centro de massa e $\vec{\omega}$ são os vetores de velocidades angulares do corpo. Todos estes vetores são representados no SRO.

E, para obter a matriz de rotação, R_i (orientação em função do tempo), pode-se utilizar a seguinte equação [1]:

$$\frac{dR}{dt} = \omega^* R$$

onde nessa equação ω^* é o dual de $\vec{\omega}$, o qual pode ser definido da seguinte forma:

$$\omega^* = \begin{bmatrix} 0 & \omega_z & -\omega_y \\ -\omega_z & 0 & \omega_x \\ \omega_y & -\omega_x & 0 \end{bmatrix}$$

4 TRATAMENTO DE COLISÕES DE CORPOS RÍGIDOS

O tratamento da colisão entre corpos rígidos envolve a detecção do ponto no qual os objetos se tocam e o cálculo da dinâmica do impulso resultante deste toque [6] e [10] . A estratégia utilizada para a colisão considera que num determinado instante de tempo existe apenas um ponto de toque entre dois objetos. Existem dois tipos de estratégias para se detectar o ponto de toque entre dois objetos [6]: o ponto de toque resultante da intersecção do vértice das arestas de um objeto com o polígono de outro objeto e o resultante da intersecção da aresta de um objeto com o polígono de um outro objeto. A análise de impacto, para resolver a dinâmica, utiliza um método analítico [9] que preserva o momento linear e o momento angular durante a colisão e resulta em uma nova velocidade linear e em uma nova velocidade angular para cada corpo rígido. Este tratamento da colisão permite ao sistema de animação realizar, em tempo de simulação, um controle automático da restrição de que dois objetos geométricos, descritos fisicamente como corpos rígidos, ao colidirem, não podem se interpenetrar.

Este método automático para controle de movimento está sendo, atualmente, integrado às rotinas desenvolvidas para análise dinâmica. Desta forma, estende-se o método de controle de movimentos para incluir reação realística a colisões.

5 PROTÓTIPO PARA ANIMAÇÃO DE CORPOS RÍGIDOS UTILIZANDO ANÁLISE DINÂMICA

O protótipo é composto de:

- Primitivas de Corpos Rígidos: coleção de corpos rígidos, tais como, esferas, cubos, paralelepípedos, pirâmides, cilindros e cones. Cada tipo de corpo define as quantidades necessárias para simulação física, tais como, centro de massa, eixos principais, momentos de inércia sobre o centro de massa e massa total.
- Pontos de Controle: pontos do corpo representado no SRO ou no SRU.
- Forças Externas: forças externas atuando no corpo. As forças externas são representadas no SRU, por ser uma forma mais intuitiva, e cada força contém uma referência para um ponto de controle em que está sendo fornecida.

A simulação é realizada a partir da descrição do estado do sistema dinâmico inicial (Figura 1). O estado dinâmico em um tempo dt mais tarde é obtido a partir do estado dinâmico inicial (onde dt corresponde ao incremento de tempo entre os quadros de animação). E, assim, cada estado subsequente é obtido a partir do estado anterior. Durante a resolução das equações de movimento são armazenados em um arquivo as novas posições e orientações dos objetos ao longo do tempo.

```
INICIO_SIMULACAO
TEMPO 0.0 0.5                /* Tempo Inicial e Tempo Final  */
INICIO_OBJETO
TIPO cubo                    /* Tipo de Corpo Rigido     */
DENSIDADE 0.0001             /* Densidade de Massa         */
POSICAO_SRU 10.0 300.0 10.0/* Posicoes do Centro de Massa   */
VEL_LIN_SRU 0.0 0.0 0.0      /* Velocidade Linear   */
FORCA_EXT_SRU -2500.0 -25000.0 10000.0 /* Forcas Ext. no Centro de Massa*/
ORIENTACAO_SRU_SRO 0.0 0.0 0.0 /* Angulos de orientacao do SRU -> SRO  */
VEL_ANG_SRO 0.0 0.0 0.0      /* Velocidades Angulares   */
PONTO_CONTR_SRO -50.0 50.0 50.0 /* Ponto de Controle*/
FORCA_EXT_SRU -24990.0 10.0 50  /* Forca Ext. no Ponto de Controle*/
FIM_OBJETO
FIM_SIMULACAO
```

Figura 1: Estado do sistema dinâmico inicial para simulação

6 RESULTADOS INICIAIS

De acordo com o exemplo de um estado dinâmico inicial apresentado na figura 1, apresenta-se na figura 2 o movimento resultante de um cubo com distribuição de massa homogênea e arestas de dimensões iguais a 100 metros sob ação de forças e torques externos caindo com a força da gravidade. O comportamento realístico deste cubo deve-se ao fato da utilização de equações dinâmicas de movimento obtidas da mecânica newtoniana.

7 CONCLUSÃO

Durante o decorrer deste trabalho, verificou-se que a estratégia adotada na dinâmica do movimento é suficientemente genérica para ser estendida com o objetivo de permitir a simulação de corpos rígidos articulados.

A geração do movimento através de equações dinâmicas baseadas em leis físicas fornece um alto grau de realismo. Porém, deve-se explorar métodos de controle que permitam adição de mais níveis de realismo e controle, tais como, detecção de colisões e respostas.

Apesar das limitações no controle das equações dinâmicas apresentadas, permitindo apenas o fornecimento da informação de estímulos iniciais (de uma forma não intuitiva para o usuário), este trabalho está atingindo seus objetivos como uma plataforma de testes

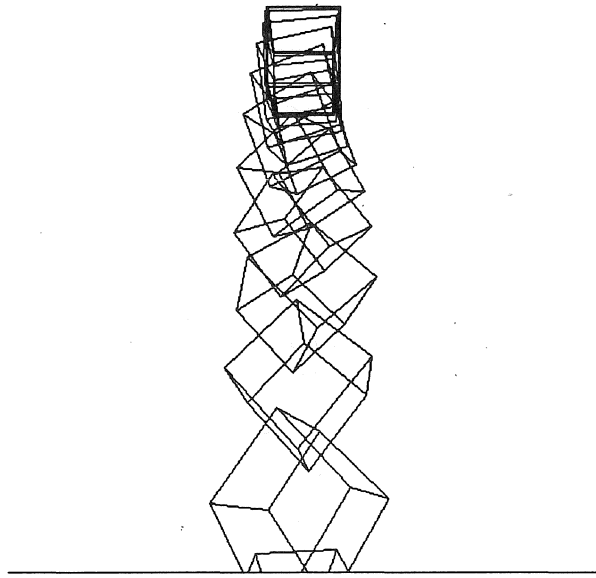


Figura 2: Cubo caindo a partir de um estado dinâmico inicial (quadros gerados através do sistema de animação PREVIEW)

para validação de um ambiente básico onde poderão ser simuladas interações entre objetos e dos mesmos com o ambiente

AGRADECIMENTOS

À professora Beatriz Gay (Instituto de Física - UFRGS) pelas conversas e esclarecimentos sobre as leis da Física, aos colegas Soraia e Rodrigo pelo auxílio junto ao PREVIEW e aos companheiros Assis e Rosana.

Referências

[1] BARR, A. H. & BARZEL, R. Controlling rigid bodies with dynamic constraints. In: Developments in physically-based modeling, New York, SIGGRAPH, August, 1988, Course Notes.

[2] FOLEY, J.; Van DAM A.; FEINER, S.; HUGHES, J. *Computer graphics principles and practice.* 2. ed., New York: Addison Wesley, 1990.

[3] FREITAS, C. M. D. S. *Técnicas de visualização em simulação.* Porto Alegre: CPGCC/UFRGS, Julho, 1990 (Trabalho Individual).

[4] GOLDSTEIN, H. *Classical mechanics.* Massachusets: Addison Wesley, 1983.

[5] GREEN, M. Using dynamics in computer animation: control and solution issues. In: *Making them Move mechanics, control, and animation of articulated figures.* 1991, p.281-314.

[6] HAHN, J. K. Realistic animation of rigid bodies. *Computer Graphics*, New York, v.22, n.4, p.299-308, Aug. 1988.

[7] LEMOS, R. R. *Um Ambiente de visualização 3D.* Porto Alegre, CPGCC/UFRGS, Abril, 1990 (Trabalho Individual).

[8] MARION, J. B. *Classical dynamics of particles and systems.* New York:Academic Press, 1970. 407p.

[9] MOORE, M & WILHELMS, J. Collision detection and response for computer animation. *Computer Graphics*, New York, v.22, n.4, pp.289-298, Aug. 1988.

[10] NETO, M. M. & BASTOS, R. M. BEZIER4D - Ambiente para modelagem, visualização e animação tridimensional. In: Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, 4., julho, 1991. *Anais*, SIBGRAPI' 91, ed. J. A. Zuffo /et. al./ São Paulo: EPUSP, 1991. XIV, 266p.

[11] PRESS, W. H.; FLANNERY, B. P.; TEUKOLSKY, S. A.; VETTERLING, W. T. *Numerical recipes in C - the art of scientific computing.* New York, Cambridge University Press, 1988.

[12] WILHELMS, J. Behavioral animation using an interactive network. In: COMPUTER ANIMATION'90, 1990, Geneva/Switzerland. *Proceedings...*, Tokio:Springer-Verlag, 1990. 237p. p.95-106.

DisCo: Um Ambiente de Programação Distribuída com Configuração Dinâmica de Processos

JORGE DE ARAÚJO LIMA FILHO
PAULO ROBERTO FREIRE CUNHA

Universidade Federal de Pernambuco
Departamento de Informática
Caixa Postal 7851
50739 Recife-PE, Brasil
jalf@di.ufpe.br
prfc@di.ufpe.br

Resumo

Flexibilidade para modificar dinamicamente a configuração de sistemas distribuídos é uma característica fundamantal para a implementação de sistemas robustos. Para atender devidamente às expectativas dos usuários, um ambiente de distribuído criado para construção de sistemas reais precisa oferecer recursos que garantam a flexibilidade, modularidade, robustez e dinânica inerentes a sistemas reais. O ambiente **DisCo** *oferece tais propriedades através das características do seu ambiente de programação e do seu ambiente de suporte à configuração dinâmica.*

Abstract

Flexibility is an important property of distributed systems. Dynamic distributed systems configuration represents a crucial point in robust systems implementation. Distributed environments have to present resources which improve the flexibility, modularity, robustness and dynamism associated with real time systems construction. **DisCo** *environment offers these properties throughout its programming and runtime supporting features.*

Key Words

Distributed Systems, Flexibility, Modularity, Dynamic Configuration.

1 Introdução

Os avanços tecnológicos ocorridos nos últimos anos nas áreas de integração de componentes(VLSI), redes de computadores e teleprocessamento viabilizaram, não apenas sob o ponto de vista técnico, mas principalmente sob o ponto de vista econômico, a interligação cada vez maior de máquinas (computadores). Inicialmente, a interligação de máquinas trouxe consigo a possibilidade de compartilhamento de recursos. Atualmente, o horizonte de aplicação de computadores interligados expandiu de tal forma que é possível não apenas o compartilhamento de recursos, mas sobretudo o desenvolvimento de aplicações distribuídas. Desta forma, não tardou a surgirem os primeiros sistemas operacionais de redes e posteriormente os primeiros sistemas distribuídos[1]. A partir daí, intensificaram-se os esforços no desenvolvimento de linguagens para programação concorrente e distribuída, ao mesmo tempo em que começaram a ser estudados os primeiros problemas relacionados com "deadlocks". Posteriormente surgiram preocupações com relação à estruturação de programas distribuídos (concorrentes) de forma a reduzir a complexidade associada com a programação de sistemas distribuídos. Isto levou à introdução de mecanismos para o fornecimento de modularidade (processos, módulos, recursos, etc) nas diversas linguagens de programação distribuída existentes. Até então, as linguagens de programação distribuída (CSP, SR, Ada, etc) permitiam a construção de sistemas distribuídos *estáticos*, ou seja, sistemas cuja estrutura permanece constante durante sua execução. Desta forma, propriedades importantes de sistemas distribuídos, como *flexibilidade* e *robustez*, não podem ser totalmente exploradas. Daí surgiram novos ambientes de programção distribuída que, explorando a flexibilidade e a robustez, permitem a construção de sistemas dinâmicos, ou seja, sistemas que podem ter sua estrutura modificada durante a sua execução. Assim como CONIC[2], Durra[3] e HPC[4], **DisCo** é um ambiente que permite e explora a configuração dinâmica na construção e evolução de sistemas distribuídos.

Os maiores problemas enfrentados pelos projetistas de ambientes distribuídos estão no fornecimento de mecanismos adeqüados para suportar a reconfiguração dinâmica de sistemas. Isto engloba desde comandos para especificar reconfigurações, até mecanismos de suporte que implementem reconfiguração dinâmica de forma segura e consistente [5]. Outro foco recente de pesquisas na área de sistemas distribuídos diz respeito ao desenvolvimento de novos modelos de gerenciamento para introdução dinâmica de mudanças em sistemas distribuídos [6, 7].

2 Ambiente DisCo

O ambiente **DisCo** foi concebido a partir de pesquisas realizadas pelo grupo de Redes de Computadores e Sistemas Distribuídos da Universidade Federal de Pernambuco (Brasil) [8, 9]. Este ambiente adota o modelo de programação a nível de configuração como metodologia para descrição, construção e evolução de sistemas.

DisCo é um ambiente para desenvolvimento de sistemas distribuídos com suporte à configuração dinâmica de processos. Este ambiente está sendo desenvolvido para permitir a modificação de sistemas de forma *dinâmica*, *segura* e *consistente*. O ambiente distribuído **DisCo** possui duas características principais que o diferencia de outros ambientes existentes [2, 3, 4], a saber:

1. afeta o mínimo possível a estrutura do sistema durante o processo de reconfiguração, mantendo assim sua máxima funcionalidade (do sistema),
2. permite a especificação de modificações na estrutura do sistema ainda na construção do mesmo para o tratamento de possíveis falhas que venham a ocorrer.

2.1 Ambiente de Programação

Sistemas distribuídos, normalmente, são complexos, não apenas pelo seu tamamho, mas sobretudo pela variedade de interações existente entre os diversos módulos que os compõem. *Modularidade* é o princípio aplicado para reduzir a complexidade associada à construção de sistemas distribuídos, onde o sistema é sub-dividido em um conjunto de tarefas simples executando paralelamente e trocando informações para implemantar a funcionalidade do sistema.

A propriedade essencial de sistemas distribuídos é a *flexibilidade*, porém muitas linguagens não permitem modelar esta propriedade de forma abrangente. Com este intuito, optamos por definir uma linguagem para programação de módulos e uma linguagem especial para definir a configuração de sistemas como um conjunto de módulos interconectados. A linguagem de configuração oferece recursos que permitem que módulos sejam inseridos e retirados dinamicamente durante a execução do sistema. Além de garantir a flexibilidade, esta solução torna a modularidade mais eficiente, pois estabelece claramente a relação entre as partes de um sistema (módulos).

O ambiente de programação de **DisCo** é formado por duas linguagens: a *Linguagem de Programação dos Módulos* e a *Linguagem de Configuração* que estabelece as conexões e permite a programação dinâmica. A utilização de linguagens distintas para a programação dos módulos e a construção do sistema facilita a descrição, compreensão e manipulação da estrutura do sistema. A construção e modificação de sistemas distribuídos é implementada traduzindo-se a sua descrição estrutural na criação e remoção de módulos e na criação, remoção e troca de conexões.

2.2 Linguagem de Programação dos Módulos

A Linguagem de Programação de Módulos[8, 10] é usada para descrever um módulo-tarefa que é uma unidade de programa sequencial[1] implementando a funcionalidade do módulo. Esta linguagem fornece *independência de contexto* permitindo a construção e compilação de módulos de software independentemente da configuração na qual eles executarão. Para isto, os comandos da Linguagem de Programação utilizados dentro de um módulo se referem unicamente à variáveis locais. Referências globais restrigem a facilidade de inclusão e remoção de módulos, além de reduzir a flexibilidade de alocar módulos a processadores. A interconexão de módulos não faz parte da Linguagem de Programação, é exclusividade da Linguagem de Configuração. Desta forma, os módulos ganham *independência de interconexão*, característica fundamental para a *reusabilidade de software.*

A comunicação entre módulos é realizada por meio de uma interface local ao módulo. Comunicação através da chamada direta a outras entidades não é permitido, pois limitaria a flexibilidade de configuração lógica de sistemas. A interface de comunicação dos módulo é formada por um conjunto de portas, que constitui o único meio de comunicação de um módulo com outros módulos e com o ambiente externo. A porta é um nome local ao módulo, desta forma todas as referências feitas por um módulo são a nomes locais a eles. Assim, garantimos a *independência sintática* dos módulos e asseguramos a reusabilidade de software. As primitivas de comunicação oferecidas pela Linguagem de Programação possuem a mesma sintaxe e a mesma semântica tanto na comunicação local (dentro de uma mesma estação), quanto na comunicação remota (entre estações). Esta propriedade, *transparência de comunicação*, facilita a construção de um sistema permitindo que este seja desenvolvido e testado em uma única estação, e depois, distribuído nas demais estações da rede. Isto é possível, porque a transparência de comunicação garante que o comportamento lógico de um módulo seja exatamente o mesmo, independentemente da estação

[1] O paralelismo ocorre a nível de módulos.

na qual irá executar. Desta maneira, o ambiente fornece mais flexibilidade no desenvolvimento de sistemas distribuídos.

2.3 Linguagem de Configuração

A Linguagem de Configuração [8, 11] é utilizada para especificar a estrutura de um sistema distribuído como um conjunto de módulos interconectados. O paralelismo acontece a nível de módulos, onde os módulos representam a unidade principal de estruturação de sistemas. Daí a necessidade de utilizarmos uma linguagem específica para manipular módulos, com mecanismos para programação dinâmica que permitam a reestruturação de sistemas para atender às mudanças do ambiente.

A especificação de um sistema distribuído consiste em quatro etapas distintas: *definição de contexto*, *instanciação*, *interconexão* e *ativação* de módulos. Na definição de contexto são especificados os tipos de módulos que constituirão o sistema. Na instanciação, especifica-se a criação das instâncias a partir dos tipos definidos na etapa anterior. Na fase de interconexão, descreve-se a maneira como as instâncias serão interconectadas e na ativação é disparada a execução das instâncias dos módulos.

A reconfiguração de um sistemas envolve não só a inclusão, mas também a remoção de módulos e conexões. A Linguagem de Configuração fornecer comandos que implementam as funções inversas àquelas apresentadas anteriormente, ou seja, *desconexão e remoção de instâncias de módulos*, *remoção de tipos de módulos do contexto do sistema* e *desativação de módulos*. A Linguagem de Configuração fornece independência entre as funções que representam cada uma das etapas de configuração. Isto é essencial para se obter maior flexibilidade na especificação de (re)configuração de sistemas. Um aspecto peculiar às linguagens de configuração usuais é a existência de uma ordem rígida para os comandos de configuração. Desta forma, todos os módulos são introduzidos de uma única vez, depois as intâncias são criadas e posteriormente são estabelecidas as ligações. Em **DisCo**, porém, esta rigidez não é aplicada, uma vez que restringiria a capacidade da linguagem, principalmente com relação à flexibilidade necessária para a construção de sistemas tolerantes a falhas.

A Linguagem de Configuração suporta abstração estrutural e decomposição modular por meio do uso de *módulos-configuração*. Um módulo-configuração é construído através da composição hierárquica de outros módulos. Para serem usados, os módulos-configuração precisam ser importados, exatamente como um módulo-tarefa. O objetivo de usar construtores distintos para módulos-configuração e módulos-tarefa é introduzir maior *clareza* ao processo de configuração de sistemas, descrevendo explicitamente a sua estrutura hierárquica. A principal diferença entre módulos-tarefa e módulos-configuração está na interface. A interface de módulos-tarefa é estática e definida por suas portas. Os módulos-configuração, por sua vez, são reconfiguráveis, portanto, suas interfaces são dinâmicas. Enquanto um módulo-tarefa é construído com os comandos da Linguagem de Programação, um módulo-configuração é construído apenas com os comandos da Linguagem de Configuração, através da composição de outros módulos em um módulo único.

A Linguagem de Configuração permite "escrever" expressões para especificar condições de configuração. Estas expressões, chamadas *expressões de configuração*, são formadas por predicados e primitivas que retornam informações sobre componentes de configuração, por exemplo: existem predicados para "checar" se uma instância está ou não ativa (`IsActive`), se duas instâncias estão conectadas (`IsConnected`), etc, e primitivas que, por exemplo, retornam onde uma determinada instância ativa está executando (`Where`).

Uma característica particular da linguagem de configuração de **DisCo** é o fornecimento de mecanismos que permitem mudar a configuração de um sistema após a execução de certos

módulos, ou seja, para que logo após a execução de um módulo um outro seja ativado. Para isto, existe um comando especial `WAIT` (*comando de reconfiguração sincronizada*) que suspende o programa de configuração até que uma dada instância termine sua execução normalmente ou seja desativada. Esta característica é de grande importância na construção de de sistemas tolerantes a falhas. Este mecanismo é geral, pois testa não apenas falhas de hardware, mas também de software. Esta característica não é encontrada em outros ambientes distribuídos como CONIC, por exemplo.

2.4 Ambiente de Execução

O ambiente de execução é constituído por um conjunto de módulos que suportam o modelo de gerenciamento de configuração dinâmica adotado em **DisCo** [9].

• Modelo de Gerenciamento de Configuração baseado em Conexões

Durante o processo de configuração dinâmica, pode haver um maior ou menor comprometimento do sistema, ou seja, pode ser muito ou pouco afetada a funcionalidade do sistema. Como pretendemos que o ambiente **DisCo** seja usado na construção de qualquer tipo de sistema, inclusive sistemas de controle em tempo real, o ideal é que o modelo de gerenciamento utilizado introduza mudanças dinamicamente afetando o mínimo possível a estrutura do sistema. Com esse objetivo, adotamos em nosso ambiente o modelo para gerenciamento de mudança de configuração baseado em conexões, que introduz modificações na estrutura de sistema dinamicamente comprometendo ao mínimo a estrutura do sistema durante o processo de reconfiguração [7, 9].

Toda e qualquer mudança de configuração afeta única e exclusivamente nós[2] e/ou conexões, que são as *entidades de configuração* de qualquer sistema . As entidades de configuração têm seus estados de aplicação mapeados em um conjunto de estados de configuração, de forma que o sistema de gerenciamento seja capaz de conduzir estas entidades para novos estados, onde a reconfiguração seja implementada de forma *segura* e *consistente*.

O estado de configuração de um nó é definido em função dos estados de configuração de suas conexões. Um nó está no estado ATIVO, quando possui ao menos uma de suas conexões ativa. Neste estado o nó opera normalmente. Quando no estado PASSIVO, o nó possui todas as suas conexões estáveis. Neste estado o nó está pronto para ser inserido ou removido do sistema.

Uma conexão no estado ATIVO opera normalmente, significando que transações podem ser iniciadas e respondidas através dela. Quando no estado ESTÁVEL, a conexão pode ser retirada do sistema com segurança. BLOQUEANDO é um estado transitório no qual uma conexão se encontra enquanto transita de ATIVO para ESTÁVEL. Neste estado uma conexão espera o término de uma transação pendente.

O protocolo de mudança é composto por um conjunto de *regras de mudança* e por um *algoritmo de mudança* que fazem com que o sistema de gerenciamento de configuração conduza as entidades afetadas pela modificação aos estados de configuração adeqüados à introdução consistente das modificações.

Regras de Mudança

Uma mudança de configuração está restrita à estrutura do sistema, e a sua especificação envolve apenas os comandos da Linguagem de Configuração: *create* e *delete* representando a criação

[2] Estamos chamando de nó um módulo em execução.

e remoção de nós, e *link* e *unlink* representando a criação e remoção de conexões. A seguir apresentamos as regras de mudança associadas com cada um destes comandos:

(*i*) CRIAÇÃO DE UMA CONEXÃO: a pré-condição para que um conexão seja criada é que o nó já tenha sido criado.

(*ii*) REMOÇÃO DE UMA CONEXÃO: a pré-condição para que um conexão seja removida é que a mesma esteja no estado estável.

(*iii*) CRIAÇÃO DE UM NÓ: a pré-condição para que um nó seja criado é sempre verdadeira.

(*iv*) REMOÇÃO DE UM NÓ: a pré-condição para que um nó seja removido é que todas as suas conexões tenham sido removidas.

Algoritmo de Mudança

Ao receber uma especificação de mudança, o sistema de gerenciamento de configuração cria, e posteriormente executa, um programa implementando a modificação especificada. Este programa, chamado de *programa de configuração*, consiste de uma seqüência de comandos gerados segundo o algoritmo de mudança apresentado a seguir.

O algoritmo de mudança é definido de acordo com as regras de mudança. À medida que o programa de configuração é executado, o sistema de aplicação é conduzido ao estado adeqüado e as modificações são implementadas. O algoritmo de mudança consiste dos seguintes passos:

1. Determina-se o conjunto de conexões a serem retiradas do sistema (*conjunto estável*).
2. Determina-se o conjunto de conexões a serem introduzidas no sistema (*conjunto de conexões*).
3. Determina-se o conjunto de nós a serem removidos do sistema (*conjunto de remoção*).
4. Determina-se o conjunto de nós a serem introduzidos no sistema (*conjunto de nós*).
5. Executa-se a seguinte seqüência de comandos de mudança:
   (a) Bloqueiam-se as conexões do conjunto estável. As conexões que possuem transações pendentes são colocadas no estado temporário BLOQUEANDO. Quando as transações pendentes terminam as conexões atingem o estado ESTÁVEL.
   (b) Neste ponto, os nós a serem removidos alcançam, automaticamente, o estado PASSIVO.
   (c) Removem-se as conexões do conjunto estável.
   (d) Removem-se os nós que fazem parte do conjunto de remoção.
   (e) Criam-se os nós do conjunto de nós.
   (f) Criam-se as conexões do conjunto de conexões.
   (g) As novas conexões são ativadas. Consequentemente, os nós criados tornam-se ativos, passando a atuar normalmente como um processo.

• Suporte de Execução

O Sistema de Gerenciamento de Configuração (SGC) é constituído por três entidades: *Gerenciador de Configuração*, *Interface de Controle* e *Servidor de Nomes*.

O Gerenciador de Configuração (GC) desempenha o papel de interface entre o SGC e o usuário (operador) do sistema de aplicação. Quando o usuário especifica uma mudança de

configuração, o Gerenciador de Configuração recebe-a, checa sua validade com respeito aos tipos de nós e portas utilizados e, se a especificação for válida, o GC gera o programa de mudança que implementará a configuração especificada. Se a especificação de mudança não for válida, o programa de mudança não é gerado e, conseqüentemente, a modificação não será implementada. Neste caso, o SGC informa ao usuário que a especificação não foi validada, para que o mesmo possa corrigi-la e submetê-la novamente.

Quando um nó é construído, o programador especifica sua *interface de aplicação* como um conjunto de portas de entrada e de saída através das quais o nó se comunica com outros nós do sistema. Além da interface de aplicação, um nó possui também uma segunda interface, chamada de *interface de controle*, que tem como função servir de ligação para que o Gerenciador de Configuração envie os comandos de mudança necessários para conduzir o nó ao estado de configuração adeqüado à introdução das modificações.

O Servidor de Nomes (SN) exerce um papel fundamental no processo de reconfiguração dinâmica, pois é nele que o Gerenciador de Configuração obtém as informações sobre o estado de configuração dos nós envolvidos na modificação.

Ao ser criado, o nó se conecta com o SN através da sua interface de controle e se registra enviando mensagens informando suas características como: nome, sistema ao qual pertence, endereço físico, estado de configuração, etc.

Desta maneira, o Servidor de Nomes tem condições de passar as informações sobre os estados dos nós para que o Gerenciador de Configuração as utilize na implementação das modificações sobre o sistema.

3 Implementação do Ambiente

A construção do ambiente **DisCo** envolve a implementação do ambiente de programação [8, 10] (Linguagem de Programação dos Módulos e Linguagem de Configuração), a implementação dos módulos do ambiente de suporte [9] (Gerenciador de Configuração, Servidor de Nomes e Interface de Controle) assim como a implementação do protocolo de mudança [7, 9], e a construção de uma interface gráfica orientada a janelas que representa o meio de interação entre o usuário e o ambiente **DisCo**.

Foi desenvolvida uma versão inicial do ambiente onde foram implementados os módulos do ambiente de suporte, o protocolo de mudança e uma versão simplificada da interface com o usuário. O ambiente de execução foi implementado utilizando-se o ambiente OPS (OCCAM Programming System) [13] por ser o único ambiente disponível com suporte à programação concorrente, na época de sua realização. As dificuldades encontradas na implementação da primeira versão de **DisCo** se deu principalmente pelo modelo de programação estático do ambiente OPS, pelas restrições de modularidade e flexibilidade apresentadas por OCCAM [14] e pela falta recursos gráficos para a construção de interfaces. Além do ambiente de suporte, implementamos também um sistema de aplicação em cima do qual foi testada a funcionalidade do ambiente **DisCo** no processo de reconfiguração dinâmica de sistemas. Apesar das limitações, esta primeira implementação do ambiente **DisCo** foi importante, pois a partir dela tivemos uma boa idéia da complexidade associada à implementação do referido ambiente.

Com a disponibilidade do ambiente UNIX juntamente com a rede de estações de trabalho Sun, dos ambientes de programação (C, C++, Sather, etc), de ferramentas para desenvolvimento de interfaces baseadas em janelas com padrão XView, e dos utilitários `lex` e `yacc` para o desenvolvimento do pré-compilador para as linguagens do ambiente de progamação, iniciamos a implementação da segunda versão do ambiente **DisCo**.

Nesta segunda versão, os módulos do ambiente de suporte estão sendo implementados em C++ [15]. A escolha de C++ se deu por dois motivos principais: compatibilidade com a biblioteca de RPC ("Remote Procedure Call") da Sun [16], que é utilizada para implementar a comunicação entre os módulos que suportam o ambiente de execução e para chamada de rotinas do sistema operacional SunOS, e facilidades de programação introduzidas pelas características do paradigma de objetos (classes, herança, sobrecarga de operadores, etc). A interface com o usuário será formada por janelas através das quais o usuário poderá verificar o estado do sistema de aplicação como um todo, de nós do sistema, de conexões de nós, o usuário poderá ter um desenho mostrando a estrutura do sistema, ou seja, a interconexão dos diversos nós que compõem o sistema, qualquer modificação introduzida no sistema é refletida no desenho de modo que o usuário tenha sempre uma informação atualizada sobre a estrutura do sistema. A interface com o usuário terá um editor específico para construção de sistemas distribuídos, de forma que os comandos sejam colocados com sua sintaxe correta e o usuário preencha os campos com os parâmetros relativos ao sistema sendo implementado, isto é verdade tanto para a Linguagens de Programação de Módulos, quanto para a Linguagem de Configuração. À medida que o ambiente for implementado, novas características poderão ser acrescentadas.

4 Conclusão

O ambiente **DisCo** foi projetado de modo a oferecer todos os recursos necessários para construção e evolução dinâmica de sistemas distribuídos. **DisCo** adota uma metodologia de programação a nível de configuração, oferecendo linguagens distintas para programação de módulos e configuração de sistemas. Dessa forma, **DisCo** fornece *clareza* na descrição de sistemas e *flexibilidade* na sua configuração. A *modularidade* fornecida pelo ambiente de programação de **DisCo** *simplifica* o processo de programação distribuída e permite a *reusabilidade* de software. O ambiente de execução utiliza um modelo de gerenciamento de configuração dinâmica que permite reconfiguração dinâmica de forma *segura, consistente* e com o *comprometimento mínimo* da funcionalidade do sistema. Os recursos que o ambiente oferece através da interface gráfica com o usuário simplificam não só a construção, mas também o controle e a evolução de sistemas distribuídos, características essenciais em qualquer ambiente que visa ser empregado no desenvolvimento de aplicações reais.

Referências

[1] *Tanenbaum, A.,S.; Renesse, R.V.*: "DISTRIBUTED OPERATING SYSTEMS", *Computer Surveys*, Vol. 17, Nº 4, Dezembro 1985.

[2] *Sloman, M.; Kramer, J.; Magee, J.*: "THE CONIC TOOLKIT FOR DISTRIBUTED SYSTEMS", *Proc. of* 6º *IFAC Distributed Computer Control System Workshop*, Monterey, Maio 1985.

[3] *Barbacci, M.R.; Weinstock, C.B.; Wing, J.M.*: "DURRA: LANGUAGE SUPORT FOR LARGE-GRAIN PARALLELISM", *Parallel Processing and Applications*, pp. 371-379, 1988.

[4] *LeBlanc, T.; Friedberg, S.*: "HPC: A MODEL OF STRUCTURE AND CHANGE ON A CONECTIVITY NETWORK", *IEEE Transaction on Computers*, C-34(12), Dezembro 1985.

[5] *Sloman, M.; Kramer, J.; Magee, J.*: "CONFIGURATION SUPPORT FOR SYSTEM DESCRIPTION, CONSTRUCTION AND EVOLUTION", *Proc. of* 5^{th} *Int. Workshop on Software Specification and Design*, Pittsburgh, Maio 1989.

[6] *Kramer, J.; Magee, J.; Young, A.*: "A Refined Model of Change Management in Distributed System", Department of Computing, Imperial College of Science and Tecnology, Agosto 1989.

[7] *Lima Filho, J.,A.; Cunha, P.,R.,F.*: "Modelo Baseado em Conexões para Gerenciamento de Configuração Dinâmica de Sistemas Distribuídos", *9º Simpósio Brasileiro de Redes de Computadores*, Santa Catarina, Maio 1991.

[8] *Justo, G.R.R.*: "Ambiente de Programação Distribuído com Configuração Dinâmica de Processos", *Tese de Mestrado*, Depto. de Informática, Universidade Federal de Pernambuco, Setembro 1988.

[9] *Lima Filho, J.,A.*: "Desenvolvimento de um Modelo Baseado em Conexões para Configuração Dinâmica de Sistemas Distribuídos", *Tese de Mestrado*, Depto. de Informática, Universidade Federal de Pernambuco, Março 1991.

[10] *Justo, G.R.R.; Cunha, P.R.F.*: "An Environment for Distributed Programming with Dynamic Configuration of Processes", *Revista Brasileira de Computação*, 5, Julho 1989.

[11] *Justo, G.R.R.; Cunha, P.R.F.*: "Programming Distributed Systems with Configuration Languages", *Configurable Distributed Systems Workshop*, Março 1992.

[13] "Occam Programming System", *STRIDE OPS User Manual*, INMOS Group of Companies, Julho 1985.

[14] *Kerridge, J.*: "Occam Programming: a Practical Approach", *Blackwell Scientific Publications*, 1987.

[15] *Stroustrup, B.*: "An Overview of C++, *ACM Sigplan Notices*, Outubro 1986.

[16] *Corbin, J.,R.*: "The Art of Distributed Applications", *Springer-Verlag*, 1991.

Programming the Asynchronous Polycyclic Architecture

Geraldo Lino de Campos[1] and Demi Getschko[2]

[1] Escola Politécnica da Universidade de São Paulo
e-mail: glcampos@pec001.usp.ansp.br

[2] Fundação de Amparo à Pesquisa do Estado de São Paulo
e-mail: demi@fpsp.fapesp.br

Abstract The Asynchronous Polycyclic Architecture (APA) is a new processor design for numerically intensive applications. One of the main features is the ability to efficiently execute loops with recurrences and conditionals. This is achieved by the use of a special iteration register, an interactive counterpart of the stack pointer for recursive operations.

Keywords: Asynchronous Polycyclic Architecture, programming, recurrences.

1. Introduction and motivation

Development of the Asynchronous Polycyclic Architecture (APA) concept was spurred by the needs of Project Omicron, an academic research effort. The project's main goal is to design and build a supercomputer for numerical applications, with a real-world performance in the same range of the then current supercomputers. APA processors are expected to provide better sustained performance in real-world problems than standard vector processors with the same peak capacity.

Our proposed Asynchronous Polycyclic Architecture [2, 3] combines the basics of the VLIW architecture [6] with the decoupled access/execute concept [8], and with extensions for loop execution, for conditional execution of instructions, for fetching large amounts of data and for an autonomous operation of groups of functional units, and eager execution for hiding

memory latency (in *eager* execution, a instruction is executed as soon as their operands are available and there is a free unit to do the operation, even when it is not sure if the control flow will warrant the need for the instruction; in *lazy* execution an instruction is executed only when reached by the control flow).

Sections 2 describes the generic APA concept in more detail. Section 3 details the polycyclic concept of the architecture. Section 4 provides a complete programming example. Section 5 offers some concluding remarks

2. Description of the APA

The Asynchronous Polycyclic Architecture resulted from a critical analysis of the characteristics of the VLIW architecture. The detailed evolution leading to the APA can be found in [2]; it will be only summarized here.

A VLIW processor is conceptually characterized by a single thread of execution, a large number of data paths and functional units, with control planned at compile time, instructions providing enough bits to control the action of every functional unit directly and independently in each cycle, operations that require a small and predictable number of cycles to execute, and each operation can be pipelined, i. e., each functional unit can initiate a new operation in each cycle.

On examining the conceptual characterization, it becomes clear that the rationale is to have a large degree of parallelism and a simple, and therefore fast, control cycle. Closer scrutiny of the conditions above shows that they are sufficient for the goal, but most are not necessary, at least in the length stated.

First, the access/execute concept was introduced. The motivation was practical considerations on memory access. General purpose architectures rely almost invariably on caches to speed up execution, exploiting the locality of memory references. Although this is the case with general computing loads, this assumption can be wildly wrong with numerically intensive programs, since dealing with large arrays is incompatible with any realistically sized cache. This conclusion is reported for quite different machines [1, 5, 7].

The APA solves the memory access problem by decoupling the process of memory access. Two kinds of functional units are used: the first, called the *address unit*, generates and sends the required addresses to the memory subsystem; the second, called the *data reference*

unit, is responsible for reordering data words coming from memory and upon request sending them to the other functional units.

Address units may operate in two modes: single address and multiple address. In the single address mode, its role is only to receive an address calculated by an arithmetic unit and to send it to the memory subsystem; in the multiple address mode, its role is to autonomously generate the values of a set of arithmetic progressions, until a specified number of elements are generated; this mode is used for reference to (a set of) arrays. Once started, the address unit can proceed asynchronously with the main flow of control.

The second step was extending this concept of asynchronous operations to the other functional units, each with its own flow of control. Experience in programming such a machine shows that it is unduly complicated. Very few situations require this full splitting of the functional units. A hierarchical system is adequate for almost all situations: the functional units can be divided into groups, composed of a certain number of arithmetic units, each capable of forking the operation of address units.

Figure 1 shows the evolution from VLIW to APA

3 Polycyclic support

Efficient execution of loops is obtained by use of a variant of polycyclic support described in [4,7]. The programming technic will be explained for a specif instance of the architecture, with the following characteristics:

- one pipelined ALU, able to perform fixed and floating operations with a latency of 5 cycles, with forwarding registers, reducing the latency to 2 cycles for the immediate use of a result;
- two memory access queues, one each for reads and writes;
- 32 general purpose registers, called *static registers*.
- 96 *dynamic registers*. The usage of these registers are described below.

3.1– Principle of operation

The dynamic registers are divided into frames, each corresponding to the temporaries needed for each iteration of a loop. The frame size is variable, and calculated by the compiler; it must be large enough to hold all the temporaries needed during the execution of one iteration of the loop. This is an efficient renaming mechanism that allows the simultaneous

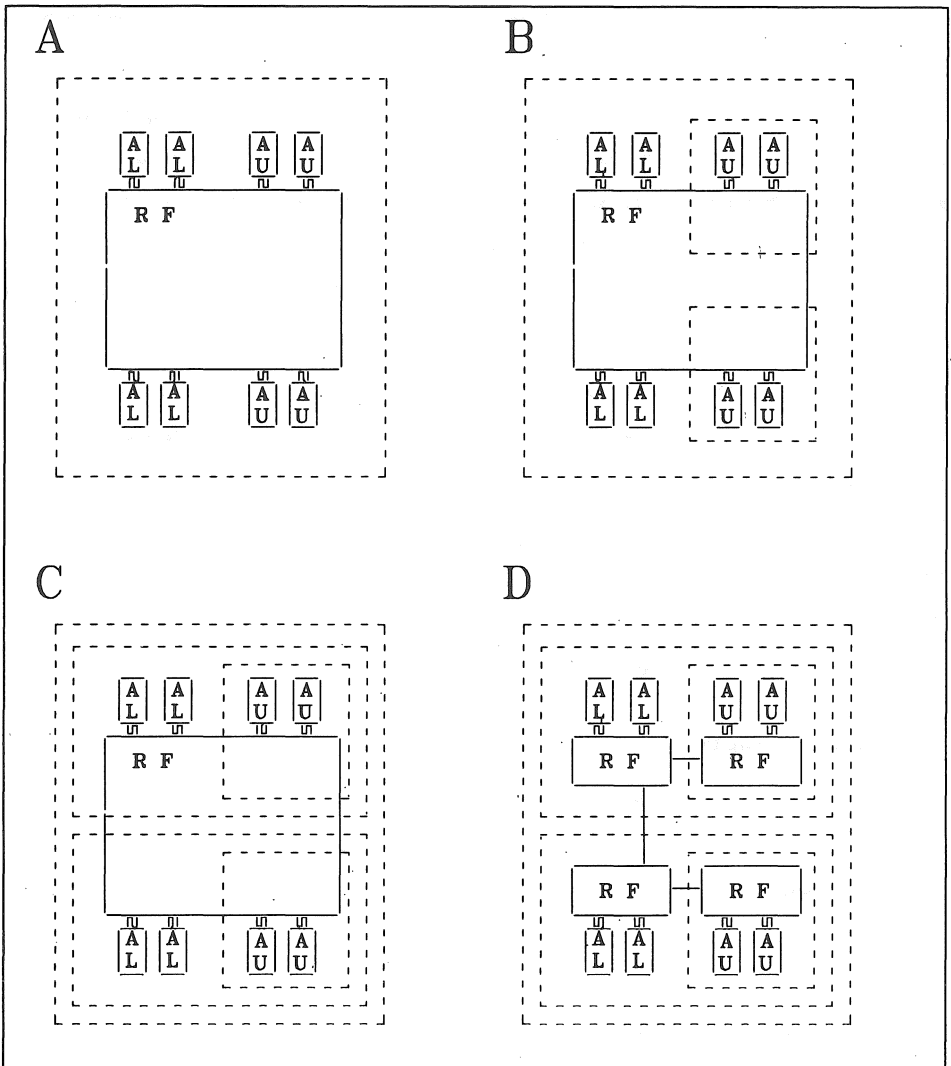


Figure 1 - Evolution from VLIW to APA architecture

In this drawing, dotted lines represent flow of control, AL stands for Arithmetic Logic Unit, and AU for Address unit.

In A, the traditional VLIW architecture. In B, evolution to the access/execute architecture. In C, the introduction of groups. Experimental analysis of programs for this architecture shows that each autonomous set may have its own copy of the register file, as shown in D. Data is exchanged via an internal bus. Other essential features for making this possible are not shown.

execution of several iterations of the same loop, hiding the ALUs latency. This is an iteractive counterpart of the stack pointer for recursive constructs.

Each frame contains the values corresponding to one iteration; old frames contains the values corresponding to older iterations. If an operation is initiated and latency would force the introduction of delays, the code can initiate a new iteration, and continue the first iteration, with values in an old frame, when executing a future iteration of the loop.

3.2– Automatic prologue and epilog

An important hardware feature is automatic prologue and epilog processing. When the execution of a loop is initiated, only the operations which refer to the current frame are executed; all the operation referring to older frames are bypassed. This phase is called prologue, and assures that only meaningful operations are done.

After the loop is executed for the expected N iterations, it is repeated M more times (this number is calculated by the compiler, and is the number of frames used by the loop). This phase is called epilogue, and the hardware operates in a complementary fashion: in the m^{th} iteration of the epilogue, only the operations that refer to frames of age m or more are executed.

This frees the compiler from generating special prologue and epilog codes, and, most important, allows it to generate code only for the general case, ignoring the special case when the loop must be executed only a few times.

4. A programming example

For the examples, the following conventions will be used:

Static registers will be represented by Rn, where n is the number of the register; dynamic register will be represented by Rn ↑ m, where n is the number of a register inside de frame, and m is the frame number, where 0 corresponds to the current frame (iteration), 1 corresponds to the values in the previous iteration, and so on.

Instructions are of the three-address format. Opcodes are self-explanatory. The source operands may be registers or the input queue, represented by Qi; the destination operand is always a register, but optionally it can be sent to the output queue, represented by Qo, as well. A flag, NEXT, indicates the end of the loop, and forces a branch to the loop's first instruction.

Before entering the loop, it is necessary to program the address units; although it is quite straightforward, it will not be shown here for space limitations. By the same reason, initialization and termination code are not shown.

As a first example, we will consider the well-known sum of products:

```
      S = 0.0
      DO 10 K = 1, N
10    S = S + A(K) * B(K)
```

The following register assignment is adopted:

R1↑ - A(K) * B(K);
R2↑ - B(K);
R5 - S

Table 1 shows the corresponding code. Version 1 is a direct compilation, without using the polycyclic features; in version 2, the NOPs following the last ADD are dropped, since the requirement of 5 cycles for reusing R5 is satisfied by the other instructions in the loop.

Version 3 uses the polycyclic feature; the ADD instruction is moved to one slot previously occupied by a NOP, and uses as operand the value of the previous iteration. As explained in the section on prologue/epilogue, the hardware does not execute this operation on the first iteration of the loop, and executes it one time more during the epilogue, so the correct number of instructions is executed.

MOV Qi, R2↑0
NOP
MUL Qi, R2↑0, R1↑0
NOP
ADD R5, R1↑0, R5
NOP
NOP
NOP
NOP
NOP, NEXT

Version 1

MOV Qi, R2↑0
NOP
MUL Qi, R2↑0, R1↑0
NOP
ADD R5, R1↑0, R5
NOP, NEXT

Version 2

MOV Qi, R2↑0
NOP
MUL Qi, R2↑0, R1↑0
ADD R5, R1↑1, R5
NOP, NEXT

Version 3

Table 1 - Code for sum of products.

In this example some NOPs are still needed to allow for the latency of 5 clock cycles; in more realistic loops, with more operations, all slots are usually filled.

To show the possibility of using loops with recurrences, lets consider the following extension to the previous loop:

```
      S = A(1) * B(1)
      DO 10 K = 2, N
      S = S + A(K) * B(K)
10    C(K) = C(K-1) + S
```

The following register assignment is adopted:

R1 ↑ - A(K) * B(K);

R2 ↑ - B(K);

R3 ↑ - S

R4 ↑ - C(K)

Table 2 shows the corresponding code. Version 1 is a direct compilation; version 2 removes the unnecessary NOPs.

Version 3 shows the polycyclic version. Another NOP slot could be used for the recurrent instruction, and each iteration still fits in five clock cycles.

5 Conclusions

The polycyclic aspect of the APA offers a simple solution for the execution of

MOV Qi, R2 ↑ 0
NOP
MUL Qi, R2 ↑ 0, R1 ↑ 0
NOP
ADD R3 ↑ 1, R1 ↑ 0, R3 ↑ 0
NOP
ADD R4 ↑ 1, R3 ↑ 0, R4 ↑ 0 [Qo]
NOP
NOP
NOP
NOP
NOP, NEXT

Version 1

MOV Qi, R2 ↑ 0
NOP
MUL Qi, R2 ↑ 0, R1 ↑ 0
NOP
ADD R3 ↑ 1, R1 ↑ 0, R3 ↑ 0
NOP
ADD R4 ↑ 1, R3 ↑ 0, R4 ↑ 0 [Qo]
NOP, NEXT

Version 2

MOV Qi, R2 ↑ 0
NOP
MUL Qi, R2 ↑ 0, R1 ↑ 0
ADD R3 ↑ 1, R1 ↑ 0, R3 ↑ 0
ADD R4 ↑ 1, R3 ↑ 0,R4 ↑ 0, NEXT

Version 3

Table 2 - Code for the second example.

loops. The increment in hardware is small, and has no impact on the cycle time.

Compared with others solutions, like software pipelining, the compiler can be much simpler. This is particularly true when considering that the automatic prologue/epilog generation allows the same code to be used even for loops of span 1, without any special consideration.

6 Bibliography

1. Abu-Sufah, W and Mahoney, A. D., "Vector Processing on the Alliant FX/8 Processor", Proc. Int'l Conf. Parallel Processing, 559-563, 1986.
2. Campos, G. L., "Asynchronous Polycyclic Architecture: an overview", Proc. of the 12th Word Computer Congress, vol I- Algorithms, Software, Architecture, Madrid, Sept 1992.
3. Campos, G. L., "Asynchronous Polycyclic Architecture", Proc. of the 6th International Conference on Parallel Processing", Lyon, Sept 1992.
4. Dehnert, J. C., Hsu, P. Y. T.., Bratt, J. P., "Overlapped Loop Support in the Cydra 5", 3rd Int. Conf. on Architectural Support for Programming Languages and Operating Systems, 26-38, April 1989
5. Diede, T., et al. "The Titan Graphics Supercomputer Architecture", IEEE Computer 21 (9):13-30, September 1988.
6. Fisher, J. A. "Very Long Instruction Word Architectures and the ELI-512", IEEE Conf. Proc. of the 10th Annual Int. Symp. on Comput. Architecture, 140-150, June 1983.
7. Rau, B. R., Yen, D. W. L. and Towle, R. A. "The Cydra 5 Departmental Supercomputer", IEEE Computer 22 (1):12-35, February 1989.
8. Smith, J. E., "Decoupled Access/Execute Architecture Computer Architectures", ACM Trans. Computer Systems, 2(4):298-308, Nov 1984.

THE USE OF A SEMI-FORMAL SPECIFICATION LANGUAGE TO BRIDGE THE GAP BETWEEN INFORMAL AND FORMAL SPECIFICATIONS OF SOFTWARE SYSTEMS: A PROPOSAL *

Stanley **LOH** (1)
José Mauro **CASTILHO** (2)

UFRGS, Instituto de Informática
Departamento de Informática Aplicada
Caixa Postal 15064
CEP 91.501 - Porto Alegre - RS, BRAZIL
E-mail: castilho@inf.ufrgs.br

(1) Prof. of Universidade Católica de Pelotas and Universidade Luterana do Brasil
(2) Prof. of Universidade Federal do Rio Grande do Sul

* This work is partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil.

ABSTRACT

The use of a semi-formal specification language in the process of constructing a formal specification for a software system is discussed. The process is executed in two separate steps, and the whole formalization activity could be performed with less effort, thus achieving better final results.

A set of heuristic rules is proposed, to guide the two translation steps (from informal to semi-formal specification, and from semi-formal to formal specification).

A dictionary of terms that appear in the specification plays an important role in the process, helping in the identification of "synonymous" parts of the specification.

Keywords: database design, conceptual modelling, software specification in logic, formal specifications.

---

1. Introduction

In the development of software systems, the phase of Requirements Analysis and System Specification is perhaps the most important one, due to its influence on the quality of the final product of the development process.

Once the specification is constructed, the results of the following design phases are in some sense consequences of this specification. If the specification is formal, we may have some advantages: non-ambiguous understanding of requirements, formal verification of properties of the specification, final user validation through the execution of the specification (with executable specifications). However, formal specifications are not so natural and legible as **informal specifications**, written

with the use of natural textual languages, or graphical and diagramatical languages of some specific application domain.

Even considering the advantages of formal languages over informal ones, some authors ([6], [1], [2], [3], [5]) advise the use of the two types of languages. In the approach taken in this work, each type of language plays a different role: informal languages are used in the requirements elicitation phase of the systems development cycle, and in the users' "visual checking" of the informations given; formal languages are useful for precisely describing requirements collected and for the unambiguous communication between the systems development agents.

Nevertheless, no matter how critical is the construction of (formal) specifications for software systems, this activity is more art then science, and depends much on the specifier's experience and personal insight on the system.

This paper introduces a two-step procedure for the construction of formal specifications for application software systems, using an intermediate, or "semi-formal", language to express the output of the first step (and input for the second). It is intended that this intermediate language has some characteristics of formal languages (it is more precise and hopefully less ambiguous than natural languages), and some characteristics of informal languages (it is more userfriendly and less complicated to use than ordinary formal languages).

Together with the semi-formal language, another essential part of the formalization procedure is a set of heuristic rules, used in the transformation steps of informal to semi-formal, and semi-formal to formal, specifications. Some of those rules (described here in natural language, to simplify the explanations), and their use, will be presented through examples, in section 3.

The approach proposed here for the construction of formal specifications of software systems is adopted in the SILOG project, an academic research project of the Information Systems Group, in Instituto de Informatica, UFRGS. The project's goal is the construction of a software environment prototype to aid database designers in their task. The Requirement Analysis and Data Modelling module of this environment accepts a set of (portuguese) natural language sentences, collected previously in interviews with users of the future database system, that form the initial informal specification.

The formal specification language used in SILOG is based on first order mathematical logic, and contains three sublanguages, one for each of the three parts that compose SILOG's formal specifications for database systems:

a. data structures definition sublanguage;
b. update and query operations definition sublanguage;
c. static and temporal integrity constraints definition

sublanguage.

The first sublanguage supports relational-like data structure definitions. The second one defines the behaviour of encapsulated operations over the database, in terms of their pre- and post-conditions. The third sublanguage is a logic language that allows the description of constraints on the data stored in the database, and of constraints on the database state transitions.

Some examples (of a library database system):

a. user(x), book(y), borrow(x,y)
b. borrowing(x,y):
   Pre-conditions: user(x) and book(y)
   Post-conditions: borrow(x,y)
c. borrow(x,y) implies (user(x) and book(y)).

In the next section we introduce the semi-formal language used in the formalization process.

2. The semi-formal language.

In the definition of SILOG's semi-formal language, some criteria served as guiding rules: sentences in the language should have simple structures. Necessarily, using such a language, one should write or speak several sentences to communicate the same idea contained in an ellaborate natural language sentence, and is induced to be more explicit and to eliminate irrelevant informations.

Another criterium used was some similarity with the formal language used in the SILOG project, with the intent to facilitate transformations from the semi-formal to formal specifications.

Corresponding heuristic rules, to guide the translation of sentences between languages, were identified. It was found necessary to have a Term Dictionary, whose main role was to ease the identification, in one specification, of those parts that had the same (or similar) meaning.

Examples:
to consult = consultation;
students = pupils;
to consult = to be consulted;

The software specification in Semi-Formal Language follows the three-part division like the formal specification, and is detailed below.

2.1 Part of Data

Informations are described with simple sentences (one verb only) in the following format:

SUBJECT VERB VERBAL_COMPLEMENT,

where SUBJECT and VERBAL_COMPLEMENT may be substantives linked

by prepositions and each substantive may be qualified by adjectives.

Examples:

users borrow books;
books have long titles.

In some cases, adverbs may be used, like in format:
SUBJECT INTRANSITIVE_VERB ADVERB.

Example: borrowings are made everyday.

2.2 Part of Operations

Each update or query operation has a name, followed by a sequence of actions. Formats are like the following:

Operation_name = VERB VERBAL_COMPLEMENT (or SUBSTANTIVE + PREPOSITION replaciong the verb);

Action = VERB VERBAL_COMPLEMENT.

Actions must be of one of the 5 types below:

- solicitation or receipt of information;
- verification of condition (in this case, an "if" could be used after the verb);
- calculation of information;
- modification of information set;
- supply of information.

Example:
Hiring employee:
Receive name and address of employee
Check if employee is not registered
Register employee.

2.3 Part of Integrity Constraints

Integrity constraints must be written according to one of the standard formats identified in [4] (total of 15 approximately).

Some examples of standard formats:
ALL/ANY subject MUST verb verbal_complement;
subject WHO/THAT/WHICH verb_1 verbal_complement_1 MAY NOT verb_2 verbal_complement_2;

Examples:
Any user must have 1 name, at most.
Users who have debts may not borrow books.

3. The Two-step Formalization Procedure, and a Sample of the Set of Heuristic Rules for Language Translation.

There are two sets of heuristic rules, each one associated to a particular translation step: (i) informal to semi-formal

translation, and (ii) semi-formal to formal translation. Each set has rules for data, operations and integrity constraints parts.

Some examples of transformations, and the corresponding rules used, are presented below in an abreviated mode (these examples were taken from a specification study of a library database system; the complete set of heuristics can be found in [4]).

3.1 Transformation from Informal Language to Semi-Formal Language

Part of Data

Informal Language:

Users borrow bibliographic materials from the library. Users must give their names when they use library services.

Semi-formal Language:

A) users borrow materials
B) users have names

Rules used:

A) - each sentence in the informal language must be analysed separately; this is a good rule!);
- verbs must be checked (some of them may have special roles, like "have", for instance);
- only transitive verbs are allowed.

B) - pronouns must be replaced by corresponding names;
- prepositions must be analysed, as well as the relationship defined by them.

Part of Operations

Informal Language:

When executing the borrowing activity, the library employee requests the user's name and the title of the desired material. After that, the employee checks if the user is registered in the library and if the material is available. If the answers are positive, the employee annotates in a special form that the user has borrowed that material.

Semi-formal Language:

Borrowing material:
Request name of the user
Request title of the material
Check if user is registered
Check if material is available
Annotate that user has borrowed material.

Rules used:

- each verb may represent an action;
- each action must be one of the 5 types established previously ;
- the order of the actions must be respected;

- physical storage information, and informations about who does the action must be disregarded;
- redundant information must be eliminated.

Part of Constraints

Informal Language:
Only faculty personnel (teachers) may borrow journals. And users may only borrow available materials.

Semi-formal Language:
A) User may only borrow journals if he/she is a teacher.
B) Only available materials may be borrowed by users.

Rules used:
A) - expressions like "must", "may not", "only", "if", "never", "always", etc, must be checked; they may identify a constraint;
- try matching the constraint with one of the defined formats.

B) - the same first rule above;
- "to be borrowed by" is the passive mode of "borrow".

3.2 Transformation from Semi-Formal Language to Formal Language

Part of Data

Semi-formal Language:
A) users borrow materials
B) users have names

Formal Language:
A) user(x), borrows(x,y), material(y)
B) user(x), have(x,y), names(y)

Heuristic rules used:
A) and B)
- transform the format "subject verb complement" to "subject(x), verb(x,y), complement(y)";
- use singular names.

Part of Operations

Semi-formal Language:
Borrowing material:
Request name of the user
Request title of the material
Check if user is registered
Check if material is available
Annotate user has borrowed material.

Formal Language:
borrowing_material(n,t):
∃ u,m (
Pre-conditions:
name(n) and title(t) and user(u) and has(u,n),
material(m) and has(m,t) and available(m).
Post-conditions:

borrows(u,m)).

Rules used:
- analyse separately each action of the operation;
- every variable should appear as argument in a "type defining" predicate;
- "input" words (words acting as objects to verbs like "to request", "to receive"), should appear as variables that are input arguments of the operation;
- preposition "of" may indicate the relation "has(x,y)";
- facts after "check if" are a pre-condition;
- expressions like "subject IS REGISTERED" must be translated to "subject(x)";
- the verb "to be" may indicate a specialization (subtype) of a concept;
- the expression "ANNOTATE fact" indicates that "fact" must be a post-condition;
- redundant occurrences must be eliminated;
- variables which appear in pre-conditions without a preceding negation symbol must be associated with an existencial quantifier ("∃").

Part of Constraints

Semi-formal Language:
A) User may only borrow journals if she/he is a teacher.
B) Only available materials may be borrowed by users.

Formal Language:
A) ∀ x,y
user(x) and journal(y) and borrows(x,y) --> teacher(x).
B) ∀ x,y
user(y) and material(x) and borrows(y,x) --> available(x).

Rules used:
A) - the format "subject MAY ONLY verb complement_1 IF HE/SHE/IT IS complement_2" is directly translated to "∀ x,y subject(x) and complement_1(y) and verb(x,y) --> complement_2(x)".
B) - the format "ONLY subject MAY verb complement" is translated to "∀ x,y complement(y) and verb(x,y) --> subject(x)";
- expressions with adjectives, like "adjective name", must be translated to "name(x) and adjective(x)";
- standard terms, defined in the Terms Dictionary, must replace their synonyms.

4. Conclusion

A semi-formal language that could be used to establish an intermediate "stepping stone" between informal and formal specifications of software systems, and a corresponding formalization procedure (based on the use of a set of heuristic language-transforming rules), were introduced in this paper.

Some small experiments were made with the use of this semi-formal language and with the set of heuristic rules, and it was noticed that the formalization process, at least for the class of applications chosen (mainly normal or "conventional" data management applications, like library management) is facilitated by the easy and simple identification of specification elements for the formal language. A bigger experiment is going on, to test more deeply our approach, in the specification of a database system for the management of the laboratories of our Instituto de Informatica.

The set of heuristic rules identified has proven to be useful for the discovery of relevant information patterns in sentences of the languages, and for the identification of transformation actions. Also, this set may be used to build a "knowledge base" for some expert system that could perform some tasks in the formalization process. However, the set is not complete yet. More experiments need to be made, in order to improve it.

Some other formalization problems remain. The approach introduced here still needs intensive participation of software specialists. Also, there is still no way of guaranteeing the "completeness" of an informal specification, or to tell that it contains all relevant information for the implementation of the system. Only end-user validation can detect missing requirements.

As a final comment, it is necessary to remember that the semi-formal language presented here was only tested with the formal language used in the SILOG Project. It is the intention of the authors to evaluate its effectiveness in the formalization process with other formal languages (like VDM, OBJ, Z, etc).

REFERENCES

[1] R. BALZER et al. Informality in program specifications. IEEE Transactions on Software Engineering, New York, v.SE-4, n.2, Mar. 1978.

[2] R. BALZER. A 15 year perspective on automatic programming. IEEE Transactions on Software Engineering, New York, v.SE-11, n.11, Nov. 1985.

[3] M. D. FRASER et al. Informal and formal requirements specification languages: bridging the gap. IEEE Trans. on Software Engineering, v.17, n.5, May 1991.

[4] S. LOH. Uma Linguagem comum entre usuários e analistas para definição de requisitos de sistemas de informação. Porto Alegre, CPGCC/UFRGS, 1991. MSc Dissertation (in portuguese).

[5] K. MIRIYALA & M. T. HARANDI. Automatic derivation of formal software specifications from informal descriptions. IEEE Trans. on Software Engineering, v.17, n.10, October 1991.

[6] T. J. TEOREY & J. P. FRY. Design of database structures. Englewood Cliffs, Prentice-Hall, 1982.

Representação de Conhecimentos em um Modelo de Dados Orientado a Objetos

Olga Maria Lopes Miranda
Lin Tse Min
Sonia Schechtman Sette
José Sergio Antunes Sette

Departamento de Informática-UFPE
Av. Prof. Luiz Freire, s/n - Cidade Universitária
Caixa Postal 7851
50.739 - Recife/PE - BRAZIL
e-mail: omlm@di.ufpe.br

Abstract

This work describes a formal approach to the expression of behavioural and deductive features in an Object-Oriented Data Model. The Object Orientation is preserved and the semantics is improved. The paradigms of Object-Orientation, Logics and Databases are integrated by formal specification to provide new resources.

Keywords

Behavioural and deductive knowledge representation, data models, object-orientation

1 Introdução

As características das aplicações ditas não convencionais estimularam a pesquisa de Modelos de Dados Orientados a Objetos (MDOOs), que se propõem a estreitar a diferença semântica entre a aplicação a ser modelada e a sua representação no modelo de dados ([3, 8, 10, 1]).

No entanto, além de permitir uma expressão mais fidedigna da realidade em questão, o Paradigma de Orientação a Objetos (POO) tornou-se de especial interesse por revelar-se como um paradigma unificador em projetos envolvendo Sistemas Baseados em Conhecimento, Bancos de Dados (BDs) e Linguagens de Programação.

Esta unificação, na realidade um desafio de enormes proporções, tem motivado a pesquisa anunciadora de uma nova geração de Sistemas Gerenciadores de Bancos de Dados (SGBDs) . Esta nova geração pretende permitir a expressão de conhecimentos procedimentais através do modelo de dados, além das características estruturais e organizacionais da aplicação modelada. Em [5] foram abordadas características requeridas por esta próxima

geração de SGBDs, que propõe "serviços" em três dimensões: gerenciamento de objetos, dados e conhecimento. Em [17] projetos de sistemas integrados que unifiquem BDs, o POO e Linguagens Lógicas para aplicações inteligentes são considerados futurísticos, ou seja, também prenunciadores de uma nova geração de SGBDs.

Este trabalho foi motivado por esta última corrente de pesquisa em BDs e compreende a especificação formal de tipos de objetos que ampliam a expressão de conhecimentos em um MDOO [14], caracterizando uma extensão do Modelo E/D [2, 15].

Na seção seguinte, a caracterização da proposta sob o enfoque de representação de conhecimentos é resumidamente apresentada. O modelo E/D é brevemente descrito na seção 3, utilizando a notação Zc [16]. Em seguida, os tipos definidos são especificados formalmente na seção 4. Finalmente, a seção 5 contém as conclusões e direções futuras.

2 Representação de Conhecimentos

Vários esquemas foram desenvolvidos para a representação do conhecimento humano de forma a obter uma definição mais natural do universo de discurso em Sistemas Baseados em Conhecimento, Sistemas Especialistas e Sistemas de Bancos de Dados Dedutivos.

Alguns destes esquemas enfatizam as características declarativas do conhecimento, outros a representação de aspectos procedimentais, enquanto outros baseiam-se nos aspectos estruturais do conhecimento a ser representado.

Sistemas de representação híbridos, que combinam as vantagens das técnicas de representação, tornaram-se então atrativos constituindo uma abordagem com alto poder de expressão de conhecimentos [9, 11].

O modelo de dados a ser apresentado pode ser classificado como um Modelo de Representação Híbrido, uma vez que: é fundamentado na filosofia de Orientação a Objetos; através do conceito de regras de produção se propõe a exprimir deduções e características comportamentais das entidades modeladas; a linguagem utilizada para a expressão de condições lógicas é baseada no Cálculo de Predicados [7].

A ênfase especial concedida ao suporte completo de diversos tipos de conhecimento permite uma modelagem precisa e natural dos aspectos do mundo da aplicação. Nesta proposta, dados e meta-dados são suportados sob um mesmo conceito: objetos. Assim, mais um passo é alcançado na unificação dos paradigmas.

3 O Modelo E/D

O Modelo Estático/Dinâmico (E/D) [2, 15] é uma extensão do Modelo Entidade/Relacionamento e caracteriza-se por ter uma visão uniforme de objetos com identificadores que são independentes de seus valores, por diferenciar entre as noções de tipos e classes e por prover uma semântica de inclusão de conjuntos, registros e seqüências para o mecanismo de generalização/especialização. Os objetos básicos da especificação do Modelo E/D [15] estão relacionados na Figura 1.

```
> GIVEN
>         NomeTipo,          // Nome de Tipo
>         NomeClasse,        // Nome de Classe
>         ValorInteiro,      // Conjunto de valores inteiros
>         ValorReal,         // Conjunto de valores reais
>         ValorString,       // Conjunto de valores string
>         INDEF,             // Valor indefinido
>         Mensagem,          // Texto
>         NomeOp,            // Nome de operação
>         IdObj,             // Identificador de objeto
>         Rótulo             // Identificador de campo de um registro
```

Figura 1

O Modelo E/D possui tipos simples pré-definidos também chamados de atômicos (Integer, Real, Bool, String) e construtores de tipos que permitem a definição de tipos estruturados. Estão disponíveis os construtores de Conjuntos (Conj), Seqüências (Seq) e Registros (Reg).

Cada tipo é caracterizado pela sua estrutura e pelas restrições que deve verificar, que são especificadas para cada um dos tipos do modelo. Além destas características próprias, cada tipo possui um nome (único entre todos os tipos que o modelo manipula), uma mensagem de descrição, os nomes dos tipos que são seus supertipos, uma indicação que informa se o tipo é excludente com algum outro tipo do modelo e um conjunto de operações que define sua interface e manipulação. Estas características comuns a todos os tipos do modelo compõem o esquema InfTipo, permitindo a modularização da especificação.

```
> SCHEMA InfTipo;
>   VAR
>       descrição      : Mensagem,
>       supertipos     : SET NomeTipo,
>       excludente     : SET NomeTipo,
>       operações      : Ops;
> END InfTipo;
```

A definição de um "Ambiente de Tipos" reúne as definições de todos os tipos que o modelo manipula em uma aplicação (tipos pré-definidos do modelo e tipos definidos pelo usuário). A identificação de cada tipo é única e corresponde ao nome do tipo. Cada tipo referenciado por outro tipo no conjunto de seus supertipos ou no conjunto dos tipos a ele excludentes, deve estar previamente definido no ambiente. Não podem existir também tipos com supertipos indicados como sendo mutuamente excludentes.

4 O Conhecimento Dedutivo e Comportamental

O Modelo E/D através do tipo Regra se propunha a tratar ações condicionais. O conceito de regras de produção ([13, 17, 11]) era utilizado, mas sem o propósito de expressar regras de dedução e reações.

Para dotar o Modelo E/D com as características requisitadas por uma nova geração de BDs, novos tipos foram definidos e outros sofreram alterações na especificação.

Através destes tipos, a capacidade de dedução e princípios de Bancos de Dados Ativos [12] são suportados no modelo. A seguir a extensão é especificada.

4.1 O tipo REGRA-DEDUÇÃO

O tipo Regra-Dedução expressa conhecimentos sob a forma de *objetos deduzidos*, constituindo a *base de dados intencional ou implícita*, enquanto os objetos persistentes constituem a *base de dados extensional ou explícita.*

A definição do tipo Regra-Dedução utilizando a notação Zc é a seguinte:

```
> SCHEMA   RegraDed;
>   INCLUDE InfTipo,
>   VAR
>       estrut : Estrutura[CorpoRegraDed],
> END   RegraDed;
```

A cláusula INCLUDE InfTipo copia todas as definições de InfTipo no esquema.

```
> TYPE
>         CorpoRegraDed   =   (sobre   : NomeEntRelaGen,
>                              se      : CondLog,
>                              entao   : SET AçãoDed);
> ALGTYPE   NomeEntRelaGen :: er NomeEntRela | erded NomeEntRelaDed;
> ALGTYPE   NomeEntRela :: ent NomeEntidade | rela NomeRela;
> ALGTYPE   NomeEntRelaDed   ::   entded NomeEntidadeDeduzida
                                | reladed NomeRelaDeduzido;
```

NomeEntidade corresponde ao nome de um tipo Entidade (entidade no ambiente de tipos). Analogamente, NomeRela, NomeEntidadeDeduzida e NomeRelaDeduzido correspondem respectivamente a nomes de tipos Rela, EntidadeDed e RelaDed no ambiente de tipos.

Assim, as características do tipo Regra-Dedução descrevem os tipos de objetos (sejam eles persistentes ou não) sobre os quais a regra está definida (aqueles presentes na parte SE), uma condição lógica a ser verificada e as ações dedutivas que devem ser executadas quando da verificação da condição.

A estrutura de uma CondLog especificada na parte SE do objeto Regra-Dedução foi inspirada naquela definida em [4]. No entanto, o POO é marcante na proposta, permitindo que condições lógicas sejam definidas sobre tipos/variáveis-objs.

A proposta aqui descrita assume que, ou uma linguagem baseada no Cálculo de Predicados [7] é fornecida pelo modelo (analogamente ao Cálculo Relacional de Tuplas no Modelo Relacional [6]) ou um mecanismo mais fraco, porém permitindo a expressão de fórmulas com um conjunto mínimo de conectivos, é suportado.

Admitindo que a primeira alternativa seja a considerada, uma condição lógica consiste em uma expressão do Cálculo de Predicados, onde os predicados são considerados os tipos que definem os objetos. Ou seja, um *predicado n-ário* representa um tipo entidade ou relacionamento com n atributos.

```
> ALGTYPE   CondLog :: ev EscopoVar | and (EscopoVar, Critério);
```

Uma CondLog compreende uma "parte de definição de escopo", que associa variáveis a

tipos de objetos e, opcionalmente, uma "subfórmula", que corresponde a uma expressão lógica sobre as variáveis-objs definidas na primeira parte.

```
> ALGTYPE  EscopoVar  ::  and (EscopoVar, EscopoVar)
>                         | and (EscopoVar, not(EscopoVar))
>                         | nerg NomeEntRelaGen (var IdObj);
```

Um EscopoVar pode ser negativo desde que haja ao menos um EscopoVar positivo na mesma regra. Isto significa que não existe no tipo o objeto especificado no EscopoVar positivo.

```
> ALGTYPE  Critério  ::  not Critério | and (Critério, Critério)
>                        | or (Critério, Critério)
>                        | aplopb (NomeOp, SEQ Expr)
>                        | aplqt (Quantif, Critério);
```

O tipo aplopb corresponde a aplicar qualquer operação de resultado booleano, de nome NomeOp à seqüência de expressões (SEQ Expr) correspondente a objetos do tipo dos parâmetros da operação e o tipo aplqt corresponde a aplicar um quantificador (universal ou existencial) a um critério. Uma expressão pode ser uma constante, uma variável ou a aplicação de operações a expressões.

Uma AçãoDed corresponde à especificação de um objeto deduzido através da definição de um predicado n-ário (correspondendo a um tipo n-ário). Ações dedutivas parametrizadas podem ser especificadas utilizando variáveis-objs (definidas na condição lógica) como argumentos das ações, sendo que uma variável pode estar associada a um tipo como um todo ou a uma propriedade de um tipo.

```
> TYPE   AçãoDed = nomeEntRelaDed (Variável);
> ALGTYPE  Variável :: var IdObj.Rótulo | var IdObj;
```

O tipo Unidade-Dedutiva consiste em uma meta-regra de dedução, no sentido em que é definido com o objetivo de agrupar objetos do tipo Regra-Dedução que definem um mesmo tipo de "objeto deduzido". Dessa forma, todos os objetos Regra-Dedução que possuírem (na parte ENTÃO do objeto) ações dedutivas a serem realizadas sobre um mesmo objeto deduzido, devem ser reunidos em um único objeto Unidade-Dedutiva.

O objeto Unidade-Dedutiva é definido como se segue:

```
> SCHEMA   UnidadeDed;
>   INCLUDE InfTipo,
>   VAR
>       estrut : Estrutura[CorpoUnidadeDed],
> END   UnidadeDed;
> TYPE   CorpoUnidadeDed =  (base      : NomeEsquemaE/D,
>                            obj_ded   : NomeEntRelaDed (SEQ Rótulo),
>                            regras    : (SEQ NomeRegraDed));
```

O esquema do Modelo E/D é especificado como o par contendo o ambiente de tipos definidos dentro da aplicação e as classes definidas para cada tipo.

O domínio de valores para cada rótulo da estrutura de um objeto deduzido não precisa ser especificado, uma vez que pode ser inferido através das regras que o definem (SEQ NomeRegraDed). As outras características dos objetos deduzidos (restrições, relacionamentos totais, etc) também são inferidas a partir dos objetos que as regras utilizam para

compor o objeto deduzido (aqueles especificados no campo "sobre" das regras utilizadas).

Para a definição de um tipo deduzido pode ser necessária a definição (também virtual) de tipos intermediários, que também devem constar na Unidade-Dedutiva relacionada.

4.1.1 A Semântica de Avaliação

A Semântica de avaliação de objetos do tipo Regra-Dedução é definida a partir da interpretação das partes SE e ENTÃO. Para a definição semântica das partes, é necessária a noção de *instanciação*.

Uma instanciação de uma Regra-Dedução consiste na substituição de cada variável livre por um objeto de uma classe associada a um tipo. Para relacionar instâncias de uma mesma classe, basta que sejam definidas variáveis-objs, quantas forem necessárias. Dessa forma, cada variável é substituída eventualmente por instâncias diferentes, conforme requerido.

A parte SE de uma Regra-Dedução retorna o conjunto de objetos que são obtidos através da instanciação das variáveis-livres presentes na condição lógica e que satisfazem aos critérios definidos. Se o conjunto vazio é retornado, a não-avaliação da parte AÇÃO da Regra-Dedução é determinada. Este conjunto só existe em tempo de avaliação da regra e é utilizado na parte ENTÃO.

Para cada instância pertencente ao conjunto resultante da avaliação da parte SE, as ações dedutivas são executadas.

4.2 O tipo REGRA-COMPORTAMENTO

A definição do tipo Regra-Comportamento utilizando a notação Zc é a seguinte:

```
> SCHEMA   RegraComp;
>   INCLUDE InfTipo,
>   VAR
>       estrut : Estrutura[CorpoRegraComp],
> END   RegraComp;
> TYPE   CorpoRegraComp   =   (se      : CondLogPer,
>                              então   : NomeAção);
> ALGTYPE   CondLogPer :: evp EscVarPer | and (EscVarPer,Critério);
> ALGTYPE   EscVarPer   ::   and (EscVarPer, EscVarPer)
>                          | and (EscVarPer, not(EscVarPer))
>                          | ner NomeEntRela (var IdObj);
```

Uma CondLogPer traduz um CondLog que não pode ser aplicada a NomeEntRelaDed, ou seja, uma RegraComp pode ser definida apenas para Entidades e Relacionamentos (persistentes).

Uma Ação corresponde a qualquer conjunto de operações especificado para objetos do tipo AÇÃO do Modelo E/D. Ações parametrizadas podem ser definidas utilizando variáveis-objs como argumentos das ações.

```
> SCHEMA   Ação;
>   INCLUDE InfTipo,
>   VAR
>       estrut : Estrutura[CorpoAção],
> END   Ação;
```

```
> TYPE  CorpoAção  =  (sobre  : SET NomeEntRela,
>                      act    : SET OpAct);

> ALGTYPE  OpAcT :: opd (OpDDL, Arg) | nop (NomeOp, Arg) | na NomeAção;
```

As operações do tipo OpDDL correspondem às operações da linguagem de definição do modelo. NomeOp corresponde ao nome das operações específicas definidas para cada tipo. Arg deve corresponder a tipos da seqüência de argumentos da operação.

```
> ALGTYPE  Arg :: var IdObj.Rótulo | cte Objeto | SEQ Arg;
```

4.2.1 O tipo REGRA-GATILHO

O tipo Regra-Gatilho contém as mesmas características do tipo Regra-Comportamento, permitindo no entanto, a definição de uma pré-condição para a execução da regra, assim como o tipo de gatilho pretendido e o estado momentâneo do mesmo.

A definição do tipo Regra-Gatilho utilizando a notação Zc é a seguinte:

```
> SCHEMA  RegraGat;
>   INCLUDE RegraComp,
>   VAR
>       estrut : Estrutura[CorpoRegraGat],
> END  RegraGat;
> TYPE  CorpoRegraGat  =  (pre-cond  : Evento,
>                          estado    : Status,
>                          tipo      : TipGat);
```

A pré-condição pode ser especificada como um evento operatório ou temporal. Um evento operatório especifica a ocorrência de operações sobre objetos ou classes, enquanto um evento temporal diz respeito à passagem do tempo. Ou seja, como em [12], um evento não se restringe a ocorrências de modificações de propriedades dos objetos manipulados no Modelo E/D. Assim, a especificação de um evento explicita quando uma determinada Regra-Gatilho deve ser avaliada.

```
> ALGTYPE  Evento :: eop EventoOp | etp EventoTp;
> TYPE  EventoOp  ::  (sobre      : NomeEntRela,
                       ocorrencia : NomeAção);
> ALGTYPE  EventoTp  ::  and (Critério, CondTemp)
                         | or (Critério, CondTemp)
                         | not (CondTemp) | ct CondTemp;
```

Uma CondTemp é uma condição que retorna um Bool e utiliza uma Classe Sistema que responde operações (mensagens) como *Datasis, DiaSis, MesSis, AnoSis.*

O estado da regra (se habilitada ou não) determina se a ocorrência da pré-condição especificada deve ou não engatilhar a regra, enquanto o tipo especifica se um gatilho é temporal ou operatório (de acordo com o evento especificado). Se o gatilho é operatório, a especificação corresponde a determinar se o mesmo deve ser avaliado antes (PRÉ) ou depois (PÓS) da ocorrência do evento.

```
> ALGTYPE  Status :: 'habilitada' | 'desabilitada';
> ALGTYPE  TipGat :: 'PRE' | 'POS' | 'TEMP';
```

O tipo Meta-Regra é o responsável pelo controle de execução de regras de comporta-

mento e gatilho, permitindo que prioridades e regras excludentes sejam expressas.

```
>  SCHEMA   MetaRegra;
>    INCLUDE InfTipo,
>    VAR
>        estrut : Estrutura[CorpoMetaRegra],
>  END   MetaRegra;
>  TYPE   CorpoMetaRegra  =  (quando        : OpAct,
>                             sobre         : NomeEn ,
>                             exclud-priori : SET (SEQ Regra));
>  ALGTYPE   Regra :: RegraComp | RegraGat;
```

Todas as regras que podem ser executadas quando uma OpAct for enviada a um determinado NomeEntRela devem ser explicitadas em "SET (SEQ Regra)", que corresponde a um conjunto de seqüências de regras. Cada seqüência dentro do conjunto determina a ordem de prioridade em que as regras devem ser executadas e diferentes seqüências indicam que a execução de uma lista é excludente à de outra.

O tipo Transação foi modificado apenas para agrupar regras de comportamento, gatilho e ações.

4.2.2 Semântica de Avaliação

Uma regra (gatilho ou comportamento) pode engatilhar a execução de outras, ocorrendo um aninhamento. Por outro lado, uma regra apenas *dispara/ativa* outra, jamais *chama* outra, ou seja, o que causa o aninhamento é a modificação de estados dos objetos.

Em objetos do tipo RegraGat, a avaliação de Evento retorna um objeto do tipo Bool. Caso o evento avalie para verdadeiro, a condição é avaliada (de modo semelhante àquele de objetos do tipo RegraDed). Se não, o par (SE, ENTÃO) não é avaliado.

Em qualquer regra (gatilho ou comportamento), se a condição se verifica, as ações são executadas (parte ENTÃO). Caso contrário, a avaliação da regra termina com a avaliação da condição.

5 Conclusões e Direções Futuras

Foram formalmente especificados os tipos que introduziram no Modelo E/D conceitos de BDs Dedutivos e Ativos compondo uma arquitetura integrada (o suporte às novas características não se deu através de camadas adicionais e interfaces, proporcionando um modelo de dados robusto e contido).

Através da definição do tipo Regra-Dedução, há uma economia de espaço de armazenamento, uma vez que os objetos deduzidos são objetos virtuais e não estão efetivamente armazenados na Base. O poder de expressão de objetos do tipo Regra-Dedução permite ainda que relações de cardinalidade arbitrária sejam definidas. Por outro lado, objetos do tipo Regra-Comportamento e Regra-Gatilho incorporaram ao modelo uma representação semântica mais fiel ao comportamento das aplicações modeladas.

Dessa forma, características de Orientação a Objetos, Lógica de Primeira Ordem e Bancos de Dados foram integradas e especificadas formalmente.

O tratamento formal ocorreu sobre as características estruturais dos tipos que permitem

a expressão de *deduções, comportamentos e reações* no modelo. A continuidade deste trabalho visa à especificação das operações destes tipos e à prototipagem da proposta, facilitada pela abordagem rigorosa do método de especificação formal utilizado.

Referências

[1] **M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier e S. Zdonik**: "THE OBJECT-ORIENTED DATABASE SYSTEM MANIFESTO", *Proceedings of the 1989, International Workshop on Object-Oriented Database Systems*, 1989.

[2] **M. Bandeira**: "MODELAGEM ESTÁTICA/DINÂMICA DE SISTEMAS DE INFORMAÇÕES", *Dissertação de Mestrado*, UFPE, Recife, 1989.

[3] **J. Banerjee, H. Chou, J. Garza, W. Kim et al**: "DATA MODEL ISSUES FOR OBJECT ORIENTED APPLICATIONS", *ACM Transactions on Office Information System*, 5(1), 1987.

[4] **M. Barthez**: "INTEGRATION D'UN LANGAGE DE REGLES DE PRODUCTION DANS UN SGBD RELATIONNEL: PRESENTATION DU LANGAGE ET DU CONTROLE DE L'EXECUTION", *These de Docteur en Informatique*, Conservatoire National des Arts et Metiers, Paris, 1990.

[5] **D. Beech, P. Bernstein, M. Brodie, M. Carey, B. Lindsay, L. Rowe, M. Stonebraker**: "THIRD-GENERATION DATABASE SYSTEM MANIFESTO". *Proc. IFIP Conference on Object-Oriented Database Systems - Analysis, Design, Construction*, 1990.

[6] **E. F. Codd**: "A RELATIONAL MODEL FOR LARGE SHARED DATA BANKS", *Communications of the ACM*, 13(6), 1970.

[7] **H.B. Enderton**: "A MATHEMATICAL INTRODUCTION TO LOGIC", Academic Press, 1972.

[8] **D. Fishman et al.**: "IRIS: AN OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEM", *ACM Transactions on Office Information Systems*, 5(1), 1987.

[9] **R. Fikes, T. Kehler**: "THE ROLE OF FRAME-BASED REPRESENTATION IN REASONING", *Communications of the ACM*, 28(9), 1985

[10] **C. Lécluse, P. Richard e F. Velez**: "O_2, AN OBJECT ORIENTED DATA MODEL", *Proc. SIGMOD International Conference*, Chicago, 1988.

[11] **N. M. Mattos**: "AN APPROACH TO KNOWLEDGE BASE MANAGEMENT", *Lecture Notes in Artificiall Intelligence, Springer-Verlag*, 1991.

[12] **C. Medeiros, P. Pfeffer**: "TRANSFORMAÇÃO DE UM BANCO DE DADOS ORIENTADO A OBJETOS EM UM BD ATIVO": *Anais do VI Simpósio Brasileiro de Bancos de Dados*, Manaus - AM, 22 a 24 de Maio de 1991.

[13] **R. N. Melo** "BANCO DE DADOS NÃO CONVENCIONAIS: A TECNOLOGIA DE BD E SUAS NOVAS ÁREAS DE APLICAÇÃO": *VI Escola de Computação*, Campinas - SP, 1988.

[14] **O. Miranda, S. Sette, J. Sette**: "ESTENDENDO UM MODELO DE DADOS ORIENTADO A OBJETOS PARA EXPRESSÃO DE CONHECIMENTO DEDUTIVO E COMPORTAMENTAL", *VII SBBD*, Porto Alegre - RS, 1992.

[15] **R. Motz**: "ESTUDO FORMAL DE UM MODELO DE DADOS ORIENTADO A OBJETOS", *Dissertação de Mestrado*, UFPE, Recife, 1990.

[16] **A. C. A. Sampaio**: "ZC: UMA NOTAÇÃO PARA ESPECIFICAÇÃO DE SISTEMAS COMPLEXOS", *Dissertação de mestrado*, UFPE, Recife, 1988. 201, 1985.

[17] **C. Zaniolo, H. Ait-Kaci, D. Beech, S. Cammarata, L. Kershberg, D. Maier**: "OBJECT ORIENTED DATABASE SYSTEMS AND KNOWLEDGE SYSTEMS", *Expert Database Systems*, 1986.

ADA REUSABLE COMPONENTS FOR THE CONSTRUCTION OF ROBUST PROGRAMS BY MEANS OF ALGEBRAIC SPECIFICATIONS OF ABSTRACT DATA TYPES *

Francis LOSAVIO
Centro de Ingeniería de Software Y Sistemas ISYS
Escuela de Computación, Facultad de Ciencias, Universidad Central de Venezuela.
Ap. 47002, Los Chaguaramos 1041-A, Caracas, Venezuela. Fax: 58-2-662-71-21
e-mail: flosavio@dino.conicit.ve

ABSTRACT. Algebraic specifications of abstract data types, taking account of genericity and exception handling, are used here as a guideline to derive reusable Ada components. This derivation is facilitated by an automatic tool which builds Ada package skeletons from the corresponding algebraic specification. The development of the Ada code considers two different levels of abstraction: the algebraic specification level, involving a specification language and the implementation level, involving the Ada language. We justify the translation of the algebraic specifications, expressed in the Pluss language, into valid Ada constructs. The fact of considering parameterized algebraic specification of abstract data types and exception handling features assure the construction of robust and reusable components, independently from the implementation language used. The major goal of this work is to show that algebraic specifications may be practically used in software development, within an assisted program construction context.

Key words: abstract data types, algebraic specifications, Ada resuable components, robust programs

1. INTRODUCTION.

The work presented in this paper is part of a major research project on systematic software construction undertaken by a work team at the Software Engineering Research Center, ISYS, Caracas, and it is aimed to study systematic application of an algebraic specification formalism, taking account of genericity and exceptional situations, for practical construction of robust and reliable software. Ada code [8] is interactively developed, within an assisted program construction framework [14], [13], from the algebraic specification of an abstract data type taking account of genericity and exception handling features. In this context, an *Ada component* is a generic package constructed from the corresponding algebraic specification. The specification language used is Pluss [10], [7], [3]. The axioms at specification levels are used by the implementor as

* This research is partially supported by the Consejo de Desarrollo Científico y Humanístico (CDCH) of the U.C.V. and the french Postgraduated Cooperation Program in Informatics of the CEFI-CONICIT.

guidelines for the implementation of the package body part and they are not directly involved in the process. The resulting Ada components are robust and generic units since we are using a formalism which takes account of exception cases and parameterized types at specification level. The features of genericity and exception handling have been determinant in the selection of Ada as the implementation language, but our approach may be similarly applied to other languages holding those features.

This work is structured into three main sections, besides this Introduction and the Conclusion. Section 2 contains the main features and building primitives of the Pluss Specification Language. As an example we present the array abstract data type specification, which will be "reused" in the next sections. Section 3 describes the mechanism followed in order to "translate" form Pluss to Ada; a complete example illustrating this process for the array data type is furnished. The structure of the proposed Ada Library is discussed. Finally in Section 4 the SPADA tool, for the partial automatization of this development process, is presented.

Some basic concepts on algebraic specifications of abstract data types will be introduced, in order to clarify the further reading.

Algebraic specifications of abstract data types. Basic concepts.

An algebraic specification [11], [12] defines a class of algebras, also called *models*, that is to say a set of operations on various sets of values. An *algebra* is just a possible *implementation* of the sorts and operation names which occur in the specification. A *data type* is characterized by one or more sets of values and by the operations performed on those values. The algebraic approach defines a data type as a *many-sorted algebra*, which is constituted by one or several sets of values, called *carriers*, and some operations defined on these sets. These sets are named, since it is necessary to distinguish them, and their names are called *sorts*. The *signature* Σ of a data type is constituted by its sorts, the names and *arity* of its operations, that is to say the sorts of their domains and co-domains. Some of the operation names are *constant*, with only a co-domain, meaning that the corresponding operation has no operands. Given a signature Σ, the many-sorted algebra, called Σ-*algebra*, is an algebra where sets and operations are named following the names of Σ. It corresponds to any implementation of the names of Σ. A Σ-*term* is any valid composition of sorted variables and operations of Σ. If the considered Σ-algebra is partial, the operations will be partial ones. We will be dealing with algebras where the operations are total ones, called E,R-algebras (Error, Recovery algebras), following the formalism of [4], which are Σ-algebras where the carriers are splitted into two disjoint sub-sets constitued by erroneous values and non-erroneous values. Exceptions and their eventual recovery are specified by means of two subsets of *declarations* . The E,R-signature is constituted by adding to Σ these two subsets. This formalism is

particularly useful because it considers exception handling, thus allowing a more realistic specification of data types. Other formalisms have been established since then in order to cope with exceptions [5], [6], but what is important is to be able to specify exceptions and not much what formalism has to be used. An *abstract data type*, which will be denoted by ADT, is a class of many-sorted algebras with the same signature and some specified common properties. In our framework, we will only consider finitely generated algebras w.r.t. the declared set of generators. An *algebraic abstract data type* is the definition of an ADT by means of a signature and a set of *axioms*. In the case of E,R-algebras, the axioms are separated into *ok-axioms* and *er-axioms*. In our context we will be using only ok-axioms and exception declarations.

2. THE PLUSS ALGEBRAIC SPECIFICATION LANGUAGE.

The Pluss (a Proposition of a Language Usable for Structured Specifications) language [10] provides a powerful way of *structuring* algebraic specifications. The simplified $ALEX_{PLUSS}$ version of Pluss [7] and [3], which includes exception handling facilities, will be used here.

2.1 The Pluss Building Primitives.

The SPEC (specifications where the class of possible implementations is fixed) are obtained by:
- an enrichment (USE keyword) , - an instance of a parameterized specification, - the fixed form of a DRAFT (specification component under design).
The BASIC SPECs are SPECs which don't enrich any other component. In this work we will be interested in SPECs components obtained by enrichment and instantiation.

2.2 Parameterization and Instantiation.

Parameterization allows the use of generic specifications, hence saves writing as many specifications as instances of a given specification are required. Parameterization involves three different entities: - a *parameterized specification* (denoted by a SPEC with the formal parameter list in brackets), - the *formal parameter specification* (keyword PAR) which include only one module. It specifies those minimal properties which have to be satisfied by the formal parameters and which have to be retained at the moment of instantiation, - the *instantiation mechanism* which consists in substituting a convenient specification (an actual parameter) to the formal parameter specification, in order to obtain the specialized version of the parameterized specification.
As an example, we will show below the generic specification of the array data type, parameterized by the INDEX and ELEM formal parameters:

```
SPEC : ARRAY [INDEX,ELEM]
SORTS : array
GENERATED BY :
init :              index index -> array
assign :            array index elem -> array
```

```
OPERATIONS :
lwb :                 array -> index
upb :                 array -> index
access :              array index -> elem
sub_array :           array index index -> array
EXCEPTION CASES :
illegal_access :            or(i<lwb(t),i<upb(t)) => access(t,i)
illegal_assign :            or(i<lwb(t),i<upb(t)) => assign(t,i,v)
illegal_init :        i>j => init(i,j)
illegal_sub_array :   or(or(i>j,i<lwb(t)),j>upb(t)) => sub_array(t,i,j)
OK-AXIOMS :
bound1:       lwb(init(i,j)) = i
... ... ...
access1:      i = j => access(assign(t,i,v),j) = v
... ... ...
sub1:         sub_array(init(i,j),k,l) = init(k,l)
... ... ...
WHERE :
t : array
i,j : index
v,v' : elem
END ARRAY
```

The INDEX formal parameter specification is the following:

```
PAR : INDEX
USE : INTEGER
SORTS : index
OPERATIONS :
first :         -> index
last :          -> index
_ : index       -> integer
EXCEPTION CASES :
first-1 : pred(first)
last+1 :        succ(last)
END INDEX
```

Note that the operation "_" is a coercion of index into integer. Thus any operation defined on integers can be applied as well on indexes (such as e.g. addition, etc.).

The ELEM formal parameter specification is expressed as follows:

```
PAR : ELEM
USE : BOOL
SORTS : elem
OPERATIONS :
eq : elem elem -> bool
OK-AXIOMS :
eq1 : eq(x,x) = true
eq2 : eq(x,y) = eq(y,x)
eq3 : and(eq(x,z) = true, eq(z,y) = true) => eq(x,y) = true
WHERE :
x,y : elem
END ELEM
```

Notice that PAR ELEM is a short specification, contained in only one module, and sufficiently general to be used as a formal parameter in other generic specifications. The specification corresponding to the data type may be seen in [15] .

3. THE ADA COMPONENTS.

In this section we will describe the mechanism involved in the development of an Ada component. We observe two abstraction levels: the specification level (the ADT is expressed in Pluss) and the implementation level (the ADT is expressed in Ada). The "translation" mechanism for passing from one level to the other and the structure of the Ada Library are described.

3.1 Translation of algebraic specifications.

We will now introduce the steps required in order to "translate" a Pluss specification into the Ada language constructs. In the following paragraphs the cases of an ordinary (non parameterized) and parameterized specification will be discussed.

3.1.1 Ordinary specifications.

We define a *translation* function ρ_S : S -> P, where S is a Pluss specification and P is an Ada package, which maps all the elements of the signature of S (sorts, operations and exception declarations) into the package specification part (data types, functions or procedures and exception declarations), corresponding to the P package in Ada. That is to say that an operation F of S, whose arity is F : TS -> TE, may be translated by ρ_S into an Ada function *f(... : in TS) return TE;* of package P. We can observe that $\rho(F)$ is not exactly equal to f, which is denoted by $\rho(F) \approx f$, because in order to obtain f we have to take into account the Ada language syntactical constructs (reserved words, special symbols, etc.). We will also assume that the elements of S will be translated by ρ_S into valid Ada identifiers. For example, in the specification of SPEC ARRAY, the sort *array* will be renamed into the *p_array* type and the specification name ARRAY will be renamed into the package identifier *sp-array*, because *array* is an Ada predefined type. Similarly, the operation *access* will be renamed into the *acces* function name identifier, because *access* is an Ada reserved word indicating an access type.

3.1.2 Parameterized specifications.

In this case, two aspects have to be considered: the translation of a generic specification into a generic Ada package and the instantiation of the corresponding formal parameters.

3.1.2.1 Translation.

This situation is similar to the non parameterized case, but two translation functions have to be defined:

1. ρ_{FPS} : FPS -> GPP, which is a mapping of the formal parameter specification FPS of a generic specification S_g into the generic part GPP of a generic package P_g.

2. ρ_{GS} : GS -> GP, which maps the signature of S_g into the generic package specification P_g.

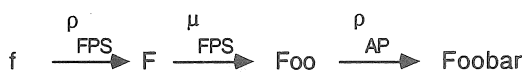
The composition of both functions ρ_{FPS} and ρ_{GS}, plus the Ada language syntactical constructs, will represent the translation of the Pluss generic specification S_g into the Ada generic package P_g, that is to say: $(\rho_{FPS}+\rho_{GS})\ (S_g) \approx P_g$.

3.1.2.2 Instantiation.

We consider, as before, the situation in which the formal parameter specifications of S_g are mapped into the generic part of the generic package P_g by function ρ_{FPS}. We have now to consider an *instantiation morphism* μ_{FPS} : FPS -> AP_{S_g} which takes the formal parameters of S_g and maps them into the corresponding actual parameters. Let us take an operation F of the FPS of S_g and the corresponding operation Foo of the actual parameters, we have $\mu_{FPS}(F)$ = Foo.

Even if there are some differences between the Pluss and Ada instantiation mechanisms, we can define a function ρ_{AP} : AP_{S_g} -> AP_{P_g}, establishing a correspondence between the actual parameters AP_{S_g} of the generic algebraic specification S_g and the actual parameters AP_{P_g} of the generic package P_g.

By means of the instantiation morphism μ_{FPS} and the functions ρ_{FPS} and ρ_{AP}, we are able to establish a correspondence between an operation of the generic part of the P_g package and an operation of its actual parameters AP_{P_g}. Taking an operation f in P_g and an operation Foobar of AP_{P_g} we will have:



3.1.2.3 An example: translation and instantiation mechanisms for the array data type.

This example illustrates in the first place the translation of the ARRAY generic specification, shown in Section 2.2, into the Ada generic package *sp_array*, by means of the translation function. In the second place, the instantiation of the formal parameters of SPEC ARRAY, by means of the instantiation morphism, will be illustrated showing also the obtention of the instantiated *sp_array* generic part.

Translation and instantiation of the parameterized specification SPEC: ARRAY.

In order to describe this process, we show below the Ada code corresponding to the specification part of the generic package *sp_array*. The algebraic specification used in the example is the SPEC ARRAY.

```
generic
  type index is (<>);
  type elem is private;
package sp_array is
  illegal_acces, illegal_assign, illegal_init, illegal_sub_array :  exception;
  type p_array is private;
function init (i,j:index) return p_array;
function lwb (t:p_array) return index;
```

```
function upb (t:p_array) return index;
procedure assign (t:in out p_array; i:index; x:elem);
function acces (t:p_array; i:index) return elem;
procedure sub_array (t:in out p_array; i,j:index);
private
  type arr_elem is array (index range <>) of elem;
  type p_array is record
                    val : arr_elem(index);
                    pri : index;
                    ult : index;
                  end record;
end sp_array;
```

We will apply function ρ_s in the first place in order to translate the algebraic specification name, the names of the exceptions and the sort into the corresponding elements of the Ada package:

ρ_s(SPEC: ARRAY) ≈ package sp_array is

ρ_s(EXCEPTION CASES :
illegal_access :
illegal_assign :
illegal_init :
illegal_sub_array :) ≈ illegal_acces, illegal_assign, illegal_init, illegal_sub_array: exception;

ρ_s(SORTS: array) ≈ type p_array is private;

Notice that, in order to clarify our discussion, we will show as domain and co-domain of the translation functions the syntactical constructions of languages Pluss and Ada.
Now, function ρ_{FPS} will be applied to the formal parameters INDEX and ELEM of SPEC ARRAY in order to obtain the generic part of the *sp_array package* .

ρ_{FPS}(INDEX, ELEM) ≈ generic type index is (<>); type elem is private;

Function ρ_{GS} is now applied to the generators and operations of the signature of SPEC: ARRAY :

ρ_{GS}(init : index index -> array) ≈ function init (i,j: index) return p_array;

ρ_{GS}(lwb : array -> index) ≈ function lwb (t: p_array) return index;

ρ_{GS}(upb : array -> index) ≈ function upb (t: p_array) return index;

ρ_{GS}(assign : array index elem -> array) ≈ procedure assign (t: in out p_array; i: index; x:elem);

ρ_{GS}(access : array index elem -> array) ≈ function acces (t: p_array; i:index) return elem;

ρ_{GS}(sub_array : array index index -> array) ≈ procedure sub_array (t: in out p_array; i,j:index);

The translation of the signature and formal parameters of the specification SPEC ARRAY is then obtained combining respectively all the applications of ρ_s , ρ_{FPS} and ρ_{GS}. We assume that $S_G = S_{G1} \cup S_{G2} \cup FPS$. For ρ_s we have : $\rho_s(S_{G1}) = \rho_s$(SPEC: ARRAY) + ρ_s(EXCEPTION CASES: illegal_access: illegal_assign: illegal_init : illegal_sub_array:) + ρ_s(SORTS: array),
and respectively for ρ_{GS}

$\rho_{GS}(S_{G2}) = \rho_{GS}$(init : index index -> array) + ρ_{GS}(lwb : array -> index) + ρ_{GS}(upb : array -> index) + ρ_{GS}(assign : array index elem -> array) + ρ_{GS}(access : array index -> elem) + ρ_{GS}(sub_array : array index index -> array)

In the case of ρ_{FPS}(FPS) we have only the application to the parameters INDEX, ELEM.

We finally obtain: $(\rho_s + \rho_{FPS} + \rho_{GS})(S_G) = P_G$

Since ρ_s, ρ_{FPS} and ρ_{GS} are bijective functions, we may apply their respective inverse functions ρ_s^{-1}, ρ_{FPS}^{-1} and ρ_{GS}^{-1} to the package P_G in order to obtain the algebraic specification S_G :

$(\rho_s^{-1} + \rho_{FPS}^{-1} + \rho_{GS}^{-1})(P_G) = S_G$

In order to describe the instantiation mechanism, we consider the same function ρ_{FPS} to map the parameters FPS into the package generic part GPP. The instantiation morphism μ_{FPS} is then applied in order to map the formal parameters FPS into the actual parameters AP:

ρ_{FPS}(INDEX, ELEM) ≈ generic type index is (<>); type elem is private;

Reciprocally we have: ρ_{FPS}^{-1}(generic type index is (<>); type elem is private) ≈ INDEX, ELEM;

We apply now the morphism μ_{FPS} to FPS in the specification S_g in order to obtain respectively sort INTER (algebraic specification for an interval) and sort INTEGER as the actual values, in the example, of parameters INDEX and ELEM : μ_{FPS}(INDEX) ≈ INTER and μ_{FPS}(ELEM) ≈ INTEGER.

Function ρ_{AP} is now used to establish the correspondence between the actual parameters of the generic algebraic specification and the actual values of the generic part of the package. Substituting the values for μ_{FPS} we obtain the required correspondence between both instantiations:

$\rho_{AP}(\mu_{FPS}$(INDEX), μ_{FPS}(ELEM)) = ρ_{AP}(INTER, INTEGER) ≈ type inter is range 1..6; package tab_int is new sp_array(index => inter, elem => integer);

Notice that the generic package *sp_array* is then instantiated by the *inter* and *integer* types, as actual values of the parameters *index* and *elem* respectively. Notice also that the name of the instantiated generic package is *tab_int*.. Since we have that $\rho_{AP}(\mu_{FPS}(FPS)) \approx AP_{P_g}$, we may automatically obtain the instantiations *inter* for *index* and *integer* for *elem* corresponding to the generic part of package P_g.

Notice that the approach followed up to now is sufficiently general, in the sense that it may applied to any implementation language, providing it holds genericity and exception handling features.

3.2 The structure of the Ada Library.

We have been inspired by the work of [9] for the structure of our Library. We will group the data types into four main classes:

1. The **building types** class. These data types allow the combination of basic types (integer, boolean, ...), in order to construct more complex data types (tree, graph, ...). We will include in this class the *array* and *record* data types.
2. The **sequential types** class. These data types are basically represented by the linear lists, containing *elements* which have to be processed sequentially. We will consider the *sequence* and *property list* data types.
3. The **tree-like types** class. These data types are constituted by sets of elements called *nodes*, hierarchically organized, with a distinguished node called *root*. The *binary tree* and *general tree* data types will represent this class.
4. The **relational types** class. These data types consist of a set of objects called nodes and relations among these objects called arcs or edges. The *graph* data type is representative of this class.

We have selected commonly used representations to implement these types. The complete Ada Library may be seen in [15]. As an example, the specification part of the array data type was shown in Section 3.1.2.3. The Ada components have been built from the corresponding Pluss specification, as it has been shown in Section 3.1. The SPADA tool is used in order to automatize the derivation of the specification part of the package and it will be described in Section 4.

4. THE DEVELOPMENT OF THE ADA COMPONENTS.

The SPADA [13] tool used for directly deriving an Ada package skeleton from its algebraic specification will be briefly described in this section. We have to mention that this tool is integrated into a prototype Ada Support Environment, system M-APEX [14], built at the ISYS research center, Caracas, which will not be detailed here. The M-APEX support environment is written in Golden Common Lisp and runs on microcomputers PC, PS or compatibles under MS-DOS. It offers several tools supporting specification and implementation as early stages of software development. Hence two different languages are supported, corresponding to the two abstraction levels of development considered: a specification language [1], [17] and a high level programming language [13]. The environment is highly interactive and the integration of the tools is seen as a way of grouping them around the abstract data type as a common or unifying concept. The tools communicate with each other consulting or updating the Knowledge Base. Internal forms are kept for the different objects manipulated within the environment.

The SPADA tool.

The SPADA tool has been inspired in the work of [18]. It generates automatically a skeleton of an Ada package directly from the corresponding Pluss specification. The skeleton is derived from the ADT signature, taking account of domains and co-domains of each operation, USEd ADT

specifications and exception declarations. SPADA produces the header and the function body skeleton for each operation. In the skeleton, the data type representation, the variables of the function header and the function bodies are missing and they have to be completed by the user. Since at Ada level the user may require some functions to be implemented as procedures for efficiency purposes, an interactive dialog is established in order to point out what are those operations that will be implemented in Ada as procedures. A small editor is also provided in order to interactively fill in the representation for the package that is being derived. The representation is actually limited to standard types and in the SPADA actual version does not allow to add type instantiations. An Ada internal form is also kept for the package. The SPADA tool also allows the integration of the function bodies which are built within the APA tool [2], in order to complete the package skeleton. In this way an executable code may be obtained (actually ADAC, the M-APEX Ada syntax driven editor [16], may be also used in order to eventually complete the type representations, instantiations and the function bodies). Nevertheless, no checkings are performed with respect to the sort and operation names. They are directly taken from the algebraic specification, in the sense that the user must take care, while specifying, not to use Ada reserved words, which may be Pluss valid identifiers.

5. CONCLUSION.

The aim of this work was to show the possibility of practical use of algebraic specifications for systematic derivation of Ada code. Moreover, the approach followed for the "translation" from the specification language to the implementation language is independent from the language used. We are not synthetizing nor prototyping programs. We are situated in the domain of assisted program construction, where the transformation of the program are mostly user-directed. Special care has been taken in the consideration of exceptions, in order to guarantee the robustness of the resulting programs. All the programs produced have been tested on an Alsys Ada compiler at the Laboratoire de Recherche en Informatique (L.R.I.), Orsay, France. Parameterized specifications make easy the reuse of the components. However, we have to point out that the algebraic specification is used only as a guide to the program construction. The axioms are not directly involved in the process. Nevertheless this fact does not minimize their fundamental importance for comprehension of the essential properties relative to the program to be constructed. The Ada Library is actually constitued by few and very well known components; we hope to extend it including more complex ones and have a choice of different representations for each component.

6. ACKNOWLEDGMENT.

The author is greatly indebted to Michel Bidoit for its always pertinent advice and fruitful remarks and to Alfredo Matteo and Françoise Schlienger, who have followed very closely this research.

7. REFERENCES.

[1] E. ACOSTA "SPEC-F: Un Formateador de Especificaciones Algebraicas expresadas en el lenguaje Pluss." Tra. esp. de Grado para la Licenciatura en Computación, UCV, Agosto 1988.

[2] L. ANES, J. HERNANDEZ "Un sistema de construcción automática de programas Ada: implementaciónde los módulos de construcción." Trab. esp. de Grado para la Licenciatura en Computación, UCV, Marzo 1991.

[3] M. BIDOIT "Pluss, un langage pour le développement de spécifications modulaires." Thèse d'Etat, Université Paris-Sud, Orsay, Mai 1989.

[4] M. BIDOIT "Algebraic specification of exception handling and error recovery by means of declarations and equations." Proceedings of the 11th. Int. Colloquium on Automata, Languages and Programming (ICALP), pp.95-108, Springer Verlag L.N.C.S. 172, 1984.

[5] G. BERNOT "Une sémantique algébrique pour une spécification différenciée des exceptions et des erreurs: application à l'implémentation et aux primitives de structuration des spécifications formelles." Thèse de 3ème cycle, Université Paris-Sud, Orsay, 1986.

[6] G. BERNOT, P. LE GALL "Label algebras and exception handling." Rapport de recherche L.R.I. Orsay, No. 712, 12/91

[7] F. CAPY "ASSPEGIQUE: un environnement d'exceptions ... ..." Thèse de 3ème cycle, Université Paris-Sud, Orsay, 1987.

[8] DEPARTMENT OF DEFENSE OF THE USA. "Reference Manual for the Ada Programming Language." ANSI/MIL-STD 15-A, January 1983.

[9] C. FROIDEVAUX, M.C. GAUDEL, M. SORIA "Types de données et algorithmes." Mac Graw Hill, Paris, 1990.

[10] M.C. GAUDEL "A first introduction to PLUSS" Rapport interne, L.R.I., Orsay 1984.

[11] J.A. GOGUEN, J. W. THATCHER, E G. WAGNER "An initial algebra approach for the specification, correctness and implementation of abstract data types." Vol. 4 of Current Trends in Programming Methodology, Prentice Hall 1978.

[12] J.V. GUTTAG "The specification and application to programming of abstract data types." Phd Thesis, University of Toronto, 1975.

[13] S. HANDAM "SPADA: Generador de un paquete Ada a partir de una especificación algebraica de un tipo de dato abstracto." Trab. esp. de Grado para la Licenciatura en Computación, UCV, Julio 1991.

[14] F. LOSAVIO "Towards the Construction of Interactive Working Environments: A prototype Ada Environment." Proceedings of the ICNTSSD'89, Caracas, November 1989.

[15] F. LOSAVIO "Dérivation systématique de programmes Ada comportant des tratements d'exception à partir de spécifications algébriques de types de données." Thèse de Doctorat, Université Paris-Sud, Orsay, Novembre 1991.

[16] Z. MILANO "ADAC: Un Editor Sintáctico Ada parta Microcomputadores tipo PC." Trabajo especial de Grado para la Licenciatura en Computación, UCV, Mayo 89.

[17] F. SANTAMARIA "Editor SPEC-E: Un Editor Sintáctico para el Lenguaje de Especificaciones Pluss-E." Trab. esp. de Grado para la Licenciatura en Computación, UCV, Junio 89.

[18] F. SCHLIENGER-DUPUY "Un environnement de programmation Ada intégrant des spécifications algébriques." Thèse de 3ème cycle, Université Paris-Sud, Orsay, Février 1984.

An Alternative Logic Programming Execution Model for Graphical Debugging

Diego Loyola
e-mail: loyola@dlrtcs.da.op.dlr.de

Abstract
The facilities provided by existing debugging tools are insufficient to give a global program execution image and do not enable a detailed program behavior analysis giving access to relevant information in a quick and easy way. In this work a Prolog execution model for debuggers that overcome these problems is presented. The developed model is based on an extended execution tree that represents successful or failed derivations. Each node of the tree has derivation information associated suitable to solve it.
The user can easily find a program mistake making use of the information provided in each node and the facilities of going directly to erroneous nodes skipping the derivation of correct or uninteresting parts.

Keywords
Logic Programming Execution Model, Trace-Debugging, Animation.

Introduction
The current work presents an execution model for the development of system debuggers where the user takes a more active position in the curse of debugging logic programs, and provides a visual feedback of the inspected program.

The computation of a logic program can be seen as a sequence of non-deterministic goal reductions or as a proof tree [1] where nodes are goals that occur in the computation and arcs represent the relation of goal invocations.

```
l1: q(X,Y) ← p(X,Y).
l2: q(X,Y) ← p(X,Z), q(Z,Y).
l3: p(a b) ← □.
l4: p(b,c) ← □.
```

Figure 1. Program example

Figure 2 shows a derivation and Figure 3 a proof tree (indentation represent the arcs) corresponding to the program example in Figure 1.

```
<q(a,Y); q(X0,Y0)←p(X0,Z0),q(Z0,Y0).; {X0→a,Y0→Y}>
<p(a,Z0),q(Z0,Y); p(a,b)←□.; {Z0→b}>
<q(b,Y); q(X2,Y2)←p(X2,Y2).; {X2→b,Y2→Y}>
<p(b,Y); p(b,c)←□.; {Y→c}>
<□, □, { }>
```

Figure 2. Example of a derivation

```
q(a,c)
        p(a,b)
        q(b,c)
                p(b,c)
```

Figure 3. Example of a proof tree

The proof tree is a more intuitive way of describing a computation, but as it can be seen, it is a less complete one. Therefore, an execution model that combines the expressiveness of proof trees and contains all the information of a derivation was developed. The model is based on an extended execution tree where each node has associated derivation information suitable to solve it. Trees corresponding to failed derivations are also obtained in order to analyze the causes that makes a goal not derivable.

Execution Model

The clauses in the program are labeled so that it can be identified which clause is used in each step of the derivation. A numeration on the tree nodes that identifies where a substitution is made and helps to find backtracking points is defined. A sign '+' or '–' is associated to each node indicating a successfully or failed goal derivation.
The behavior of the standard Prolog systems is obtained fixing the computation and search rules such that it is selected the leftmost atom in a goal making a depth–first travel.

The formal definition of the execution model is as follows:

A *labeled clause* is a universally quantified logical sentence of the form

$$L:\ A \leftarrow B_1, \ldots, B_k. \qquad k >= 0.$$

where L is the clause unique identifier, the A and the Bs are logical atoms. A is the clause head and the Bs the body.

A *goal* is a sentence of the form

$$\leftarrow B_1, \ldots, B_k \qquad k >= 0.$$

A *labeled logic program* is a finite set of labeled clauses.

The *extended execution tree* τ of an unique atom goal G from a labeled logic program P, node number N in N^+ and proof of sign S in $\{+, -\}$ is defined as:
(a) Let l : $A \leftarrow \square$. be a labeled clause in P such that A and G are unifiable via a mgu θ. The extended execution tree of G from P clause l, node number N and sign '+' is

$$\tau_{G\theta l N+} = \langle G\theta, \theta, l, N, +, \{ \langle \text{success}, \phi, l_s, N_s, +, \{\ \} \rangle \} \rangle .$$

where l_s and success do not appear in P, ϕ is the empty substitution and $N_s = N.0$.

(b) Let $l: A \leftarrow B_1, \ldots, B_k.\ k > 0$ be a labeled clause in P such that A and G are unifiable via a mgu θ. The extended execution tree of G from P clause l, node number N and sign S is

$$\tau_{G\alpha lNS} = \langle G\alpha, \delta, l, N, S, \{\tau_{C_1\theta_1 1_1 N_1 S_1}, \ldots, \tau_{C_k\theta_k 1_k N_k S_k}\}\rangle .$$

where $C_1 = B_1\theta$, $C_i = B_i\theta\theta_1 \ldots \theta_{i-1}$ for $i = 2,\ldots,k$, $\delta = \theta\theta_1 \ldots \theta_k$, $\alpha = \delta$ restricted to the variables that appear in G, $Ni = N.i$ for $i = 1,\ldots,k$ and S is '+' if S_i is '+' for $i = 1,\ldots,k$ or '−' otherwise.

(c) If for all labeled clauses A in P, A and G can not be unified, the extended execution tree of G from P clause l_f, node number N and sign '−' is

$$\tau_{G\phi l N-} = \langle G, \phi, l, N, -, \{\langle fail, \phi, l_f, N_f, -, \{\ \}\rangle\}\rangle .$$

where l_f and fail do not appear in P, ϕ is the empty substitution and $N_f = N.0$.

NOTE: A multiple atom goal $\leftarrow G_1, \ldots, G_k$. can be defined as the labeled clause $l_j: G \leftarrow G_1, \ldots, G_k$. where l_j and G do not appear in P.

In figure 4 the extended execution tree of q(a,Y) for the program in Figure 1 and root node number 1 is shown.

```
<q(a,Y){Y → c}, {X0→a,Y0→Y},l2,1,+,{
      <p(a,Z0){Z0 → b},{ },l3,1.1,+,{
            <success,{ },ls,1.1.0,+,{ }>}>
      <q(b,Y){Y→c},{X2→b,Y2→Y},l1,1.2,+,{
            <p(b,Y){Y→c},{ },l4,1.2.1,+,{
                  <success,{},ls,1.2.1.0,+,{ }>}>}>
```

Figure 4. Example of a extended execution tree

The enriched tree represents successful or failed derivations and has a node for each goal generated by the computation. Additional information is associated to the node: the clause used to solve the goal, variable bindings before and after the derivation and information on whether the node derivation was successful or not.

Debugging

The enriched tree represents successful or failed derivations and has a node for each goal generated by the computation. The following information is associated to the nodes: clause used to solve the goal, variable bindings before and after the derivation (substitution), a numeration and sign of the node derivation. The two debugging alternatives based on the defined execution model were explored: stepwise and graphical debugging.

Stepwise Debugging

The tree is stepwise displayed using menus and the navigation is made for levels, i.e. when a node p is selected, the system shows it and its direct descendents $p_1, \ldots, p_k$. The goals are pressed by '+' or '–' sign if their respective proofs are success or fail. See figure 5. In each moment the user can decide to see the information associated to p (zoom), to go to next level selecting a p_i i=1...k or to return to the previous level. Finally the user has the possibility of analyzing the next extended execution tree.

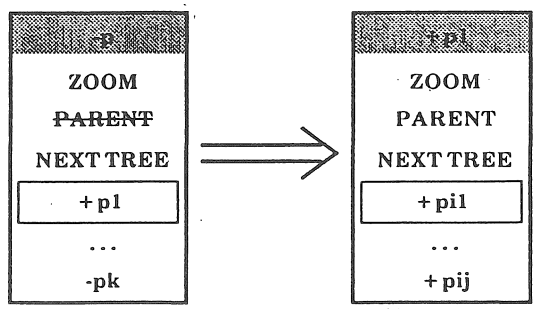


Figure 5. Stepwise Debugging

The developed execution model was implemented using the meta-level reasoning facilities provided for the existing logic languages. A meta-interpreter computes the extended execution tree while solving a goal.

Graphical Debugging

The tree is shown graphically without any additional work from the user. Nodes that can not be proved are marked with a color darker than that of those that could be derived. The user can select any node in order to examine it.

Figure 6 shows a failed derivation of the goal q(a,Y) in the program shown in Figure 1.

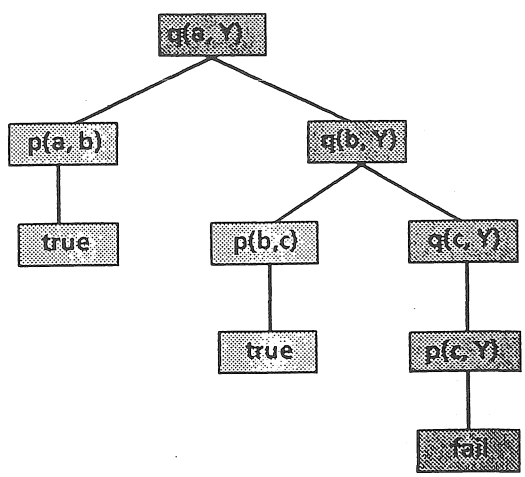


Figure 6. Animation and Graphical Debugging

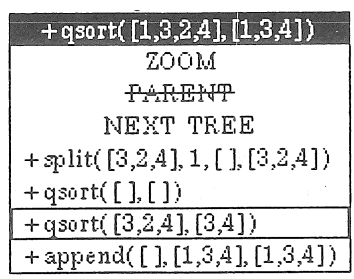
The navigation in the tree is made with commands (key strokes) that change the "*current*" node, moving to nodes in the same level or in different ones. There are scrolling facilities that enables the navigation in "big" trees. In each node the user can ask the system for additional information (zooming).
This graphical debugger is part of a pure Prolog environment based in the developed execution model.

Debugging example
This navigation facilities and the information associated to each node allows the user to have a clear idea of the actual program behavior, and to identify and reach directly the suspicious nodes. In the following example the develped tools [3] are used to find the problem in the buggy quicksort shown in Figure 7

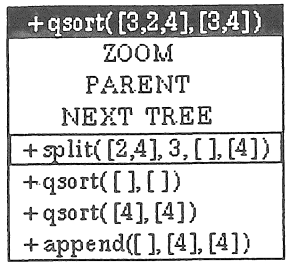
```
qsort( [X | Xs], Result)  ←
          split(Xs, X, Lo, Hi),
          qsort(Lo, ResultLo),
          qsort(Hi, ResultHi),
          append(ResultLo, [X | ResultHi], Result).
qsort( [ ], [ ]).
split( [X | Xs], Crit, Lo, [X | Hi])  ←
          X >= Crit,
          split(Xs, Crit, Lo, Hi).
split( [X | Xs], Crit, Lo, Hi)  ←
          X < Crit,
          split(Xs, Crit, Lo, Hi).
split( [ ], Crit, [ ], [ ]).
append( [X | Xs], Ys, [X | Zs])  ←
          append(Xs, Ys, Zs).
append( [ ], Ys,Ys).
```

Figure 7. Buggy quicksort

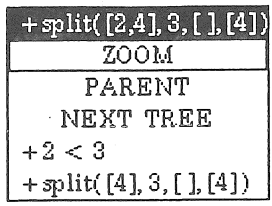
With the goal qsort([1,3,2,4],X) the system displays the following menu



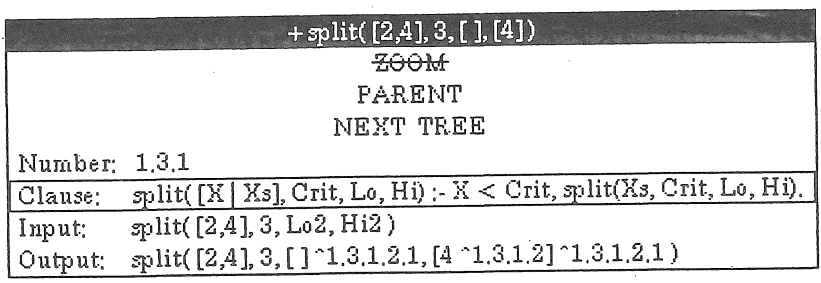
The root goal is solved with an erroneous result X=[1,3,4], the element "2" is lost. The nodes used in its derivation +split([3,2,4],1,[ ],[3,2,4]) and +qsort([ ],[ ]) are correct. The second element in +qsort([3,2,4],[3,4]) is lost. Selecting this qsort node we obtain the derivation



The element is lost in +split([2,4],3,[ ],[4]). Its derivation is



The nodes +2<3 and +split([4],3,[ ],[4]) are correct ... the problem must be in +split([2,4],3,[ ],[4]) self. Zooming this node



The mistake is detected: the element "X" in the clause used to solve the predicate split is not added to "Lo" as it should be.
In this example, the user has had to direct the debugger only three times to get the error. More user interactions and time are needed in order to find the bug using the traditional debugging methods.

Conclusions

The proposed execution model enables a debugging methodology that solves the main problems found in the traditional trace packages (for example the procedure box model) and it is more intuitive that the TPM model [2].

A detailed analysis of Prolog program behavior is provided using the extended execution tree. The developed tool leaves the notion of desirability of program behavior up to the user during debugging. He/she decides which are the "suspected" nodes and can directly reach them.

Stepwise and graphical debugging alternatives were developed. The information provided in each node and a flexible navigation through the extended tree enables the user to go directly to erroneous nodes skipping the derivation of correct or uninteresting parts, and to drive the debugging process only a few times as was verify in different test sessions.
The main facilities provided by the developed tool are the possibility of:

- Giving a visual feedback of program execution.
- Having a flexible navigation through the tree.
- Knowing wether the node derivation was successful or not.
- Identifying the clause used to solve a goal.
- Knowing variable binding before and after the derivation.
- Identifying the node where a variable is instantiated.

The visualization of the execution tree can also be useful for educational purposes and as a tool for Prolog programs debugging.

Future work will be concern with the graphical representation of "big" trees (giving a global view while enabling to concentrate the attention in a specific subtree) and an automatically empirical analysis of the extended execution tree looking for 'bug patterns' that identify the possible cause of the program failure.

Bibliography:

[1] JW. Lloyd (ed), Computational logic, Brussels; spring–verlag; 1990
[2] H. Eisenstadt, The Transparent PROLOG Machine (TPM): an execution model and graphical debugger for logic programming, Journal of Logic Programming, Vol. 5 No. 4 pages 277–342, 1989
[3] D. Loyola, Extending Execution Trees for Debugging and Animation in Logic Programming, Lecture Notes in Computer Science, No. 528, 1991

Um Algoritmo para a Unificação na Lógica de Ordem Superior

Artêmio Ludwig
Departamento de Informática
Universidade Federal de Viçosa
36570-000 Viçosa (MG) – Brasil

Wagner C. do Amaral
Faculdade de Engenharia Elétrica – C.P. 6101
Universidade Estadual de Campinas – UNICAMP
13081-000 Campinas (SP) – Brasil

20 julho 1992

Resumo **A implementação de interpretadores para a linguagem da Lógica de Ordem Superior (LOS) ainda conta com dois desafios: (a) tornar a linguagem mais *amigável* permitindo estimular sua adoção e (b) implementar interpretadores cujo desempenho não sacrifique sua usabilidade.**

Este trabalho é resultado da prototipação de um interpretador para a linguagem da LOS chamado λ-Prolog, no qual são tomadas apenas as sentenças definidas positivas. Nela a unificação é um dos procedimentos mais importantes e apresenta complexidade elevada. Seu processamento é razão de depauperação na qualidade do sistema. Partindo-se dos algoritmos SIMPL e MATCH já conhecidos, propõe-se um algoritmo alternativo para MATCH com indicações de melhor desempenho e com uma interpretação mais facilitada do mecanismo da unificação.

Palavras-Chave: Unificação, Match, Lógica de Ordem Superior.

1 Introdução

Excluindo a parte de inicialização, que inclui a conferência gramatical das sentenças, a formação da base de conhecimento e a tipificação de constantes e variáveis, a implementação do interpretador para a LOS tem procedimentos similares àqueles do interpretador para a primeira ordem. Algumas partes são específicas deste sistema. Evidenciam-se dois aspectos:

1. A unificação que na LOS não apresenta um *mgu*.
2. O retroencadeamento conta com mais uma razão para acontecer.

A complexidade da unificação constitui-se num desestímulo para avanços no uso da linguagem da LOS, quer por falta de seu entendimento quer por suas dificuldades computacionais.

Aborda-se, na sequência , o algoritmo da unificação especificado em [1] e [4] , (Fig.1) apresentado com mais detalhes por [1]. Os mesmos autores formalizam a linguagem sobre a

*Trabalho parcialmente financiado pela Fundação de Amparo à Pesquisa de Minas Gerais.

qual o algoritmo foi proposto e trabalha-se com a notação e as fórmulas definidas positivas especificadas em [4]. Em particular busca-se analisar a *imitação* e a *projeção*. Espera-se que o leitor esteja familiarizado com a terminologia usada.

Para o estudo de MATCH e a proposição de novos algoritmos considera-se que o primeiro par da esquerda do conjunto discórdia sempre é tomado para dar sequência ao processo de unificação. Adota-se a caracterização de "unificador parcial" para a composição dos unificadores obtidos até determinado nó da árvore de unificação, chamado ainda de unificador para o ramo. O unificador que permite a passagem de um nó para outro é chamado de unificador do arco. Usa-se ainda os símbolos $\bar{w}$ e $\bar{x}$ para representar as sequências $w_1 \cdots wr$ e $x_1 \cdots x_n$, respectivamente.

Na próxima seção são descritas algumas características do algoritmo MATCH. Três lemas são enunciados e constituem a base para o método de obtenção dos unificadores "principais" que são assunto para a seção 3. A seção 4 apresenta a combinação de unificadores que é uma proposta para se obter os demais unificadores da árvore de unificação de MATCH. Os resultados de algumas aplicações dos algoritmos propostos são vistos na seção 5 e as conclusões encontram-se na seção 6.

2 Características Gerais

Considera-se $\langle F_1, F_2 \rangle$ um par de discórdia com duas fórmulas de mesmo tipo, uma flexível e outra rígida assim descrito:

$$\langle F_1, F_2 \rangle = \langle\, f\, A_1 \cdots A_r \,,\, C\, B_1 \cdots B_s \rangle$$

Os três lemas que se seguem, cujas demonstrações podem ser vistas em [2], usam como suporte a árvore de unificação gerada por MATCH. Os algoritmos SIMPL e MATCH extraídos de [3] estão colocados na Fig. 1.

Lema 2.1 *Uma sequência de imitações aplicadas a qualquer par de discórdia*

$$\langle f\, A_1 \cdots A_r, C\, B_1 \cdots B_s \rangle$$

produz um ramo de sucesso cujo unificador parcial é da forma

$$\theta_n = f/\lambda \bar{w}\, C\, B_1 \cdots B_n\, (h_{n+1}\bar{w}) \cdots (h_s\bar{w})$$

e o número total de arcos np(f) para atingir este nó da árvore de unificação é definido por:

- $np(f) = 1$ *se* $r = 0$,
- $np(f) = 1 + \sum_{i=1}^{r} np(A_i)$ *se* $r > 0$.

O lema não considera os pares flexível-flexível que podem surgir na árvore de unificação. O algoritmo MATCH está proposto apenas para pares flexível-rígido e seu resultado consiste de um unificador onde o conjunto D pode ser vazio, F (de falso , caso em que não houve unificação) ou contém apenas pares flexível- flexível. Nosso método obterá sempre como resultado um unificador com $D = \emptyset$ ou $D = F$.

Na projeção, segundo o algoritmo MATCH, quando um dos argumentos de F_1 é funcional,

Definição da Rotina SIMPL: Seja o conjunto de discórdia D constituido de pares de fórmulas em sua forma normal de unificação (FNU).

1) Se D = $\emptyset$ então SIMPL(D) = $\emptyset$.

2) Se D = $\{\langle F_1, F_2\rangle\}$ e
   a) F_1 é flexível então SIMPL(D) = D; caso contrário
   b) Se F_2 é flexível então SIMPL(D) = $\{\langle F_2, F_1\rangle\}$;
   c) De outro modo F_1 e F_2 são ambos rígidos. Sejam $\lambda\bar{x}(C^1A^1\cdots A^r)$ e $\lambda\bar{x}(C^2B^1\cdots B^s)$ as formas normais de unificação para F_1 e F_2. Se $C^1 \neq C^2$ SIMPL(D) = **F**, caso contrario SIMPL(D) = SIMPL($\{\langle\lambda\bar{x}.A^i, \lambda\bar{x}.B^i\rangle$, $1 \leq i \leq r\}$)

3) De outro modo D tem no mínimo dois membros. Seja D = $\{\langle F^i, G^i\rangle\}$, $1 \leq i \leq n\}$,
   a) Se SIMPL($\{\langle F^i, G^i\rangle\}$) = **F** para algum i então SIMPL(D) = **F**
   b) Senão SIMPL(D) = $\bigcup_{i=1}^{n}$ SIMPL($\{\langle F^i, G^i\rangle\}$).

Definição da rotina MATCH: Sejam V um conjunto de variáveis, F_1 e F_2 duas fórmulas na FNU $\lambda\bar{x}(f\,A^1\cdots A^r)$ e $\lambda\bar{x}(C\,B^1\cdots B^s)$, de mesmo tipo, flexível e rígida, respectivamente. Além disto, seja $\alpha_1 \rightarrow \cdots \rightarrow \alpha_r \rightarrow \beta$ o tipo de f e para $1 \leq i \leq r$, seja w_i uma variável do tipo α_i

i) Se C é uma variável (isto é C aparece em $\bar{x}$), então IMIT(F_1, F_2, V) = $\emptyset$; Caso contrário (C é constante) sejam $h^i \notin$ V $\cup$ $\{\bar{w}\}$ variáveis de tipos adequados para $1 \leq i \leq s$, isto é, do mesmo tipo de $B^1\cdots B^s$ então IMIT(F_1, F_2, V) = $\{\langle f, \lambda\bar{w}(C(h^1\bar{w})\cdots(h^t\bar{w}))\rangle\}$.

ii) Para $1 \leq i \leq r$, se α_i não for da forma $\beta_1 \rightarrow \cdots \rightarrow \beta_t \rightarrow \beta$ então $PROJ_i(F_1, F_2,$ V) = $\emptyset$; Caso contrário, sejam $h^i \notin V \cup \{\bar{w}\}$ variáveis de tipos adequados para $1 \leq i \leq t$, então $PROJ_i(F_1, F_2, V) = \{\langle f, \lambda\bar{w}(w_i\,(h^1\,\bar{w})\cdots(h^t\,\bar{w}))\rangle\}$.

iii) MATCH(F_1, F_2, V) = IMIT(F_1, F_2, V) $\cup$ $\bigcup_{1\leq i\leq r} PROJ_i(F_1, F_2, V)$

Figura 1: Definição das rotinas SIMPL e MATCH [3]

o unificador gerado para o arco é

$$\lambda\bar{w}.w_i(h_1^i\bar{w}\cdots h_t^i\bar{w})$$

onde t é o número de variáveis ligadas por λ no argumento A_i projetado.

Este unificador existirá se houver unificação entre A_i aplicado a

$$\xi = ((f/h_1^k)F_1, (f/h_2^k)F_1, \cdots, (f/h_t^k)F_1)$$

e F_2, isto é, A_i aplicado a uma série de t argumentos $(f/h_i^k)F_1$, tantos quantas forem as variáveis de ligação de F_1. A expressão $(f/h_i^k)F_1$ representa F_1 com f substituido por h_i^k. Doravante representa-se esta aplicação por $\xi(A_i)$. Para argumentos A_i não funcionais, não haverá variáveis de ligação e $\xi(A_i) = A_i$.

Sujeito pois a condição de que exista um ou mais unificadores para $\langle \xi(A_i), F_2 \rangle$ o unificador parcial inicial obtido por projeção é então

$$\{\ f/\lambda\bar{w}\ C[\ w_i(h_1^i w_1 \cdots w_o) \cdots (h_t^1 w_1 \cdots w_o)\ ]\ \}.$$

representando a projeção do i-ésimo argumento de f.

Lema 2.2 *Uma projeção de A_i de F_1 sobre F_2, após imitações sucessivas sobre os k-1 A_i anteriores, ($k \geq 2$), gera unificadores parciais que têm a forma:*

$$\{\ \phi(f/\lambda\bar{w}\ C\ B_1 \cdots B_{k-1}\ (w_i\ (h_1^k\bar{w}) \cdots (h_t^k\bar{w}) \cdots (h_s\bar{w})) \cup \phi\ \},$$

em que $\phi = unif(\langle \xi(A_i), B_k \rangle)$, unif representa um dos conjuntos de unificadores para o par, k a posição que $(w_i(h_1^k\bar{w}) \cdots (h_t^k\bar{w}))$ ocupou como argumento na substituição para f e t, o número de variáveis ligadas de B_k, não iguais à sua cabeça.

A segunda parte do conjunto pode ser vazia ou ser um conjunto de substituições, indicando se houve sucesso na projeção.

Lema 2.3 *Qualquer unificador parcial θ_n, obtido por meio do uso dos lemas 1 e 2, depois de obtidos os unificadores ϕ, pode ser transformado em um unificador de $\langle F_1, F_2 \rangle$ pela substituição de cada termo cuja cabeça é h_i, ($1 \leq i \leq r$) existente no unificador parcial por seu correspondente $\theta_n(B_i)$. O termo $\theta_n(B_i)$ representa a aplicação do unificador parcial a B_i.*

Este lema é consequência da possibilidade de se efetuar tantas imitações quantas necessário após qualquer nó, até que não haja mais termos h_i.

3 Otenção dos Unificadores "Principais"

Como consequência dos lemas anteriores pode-se ter, a partir da primeira imitação, unificadores para f que são a fórmula F_2 abstraida de $\lambda\bar{w}$ com qualquer $w_i[\cdots]$ substituindo um dos argumentos de F_2.

Método similar ao usado para derivar os lemas 2.1, 2.2 e 2.3 permite mostrar que os termos $w_i[\cdots]$ podem substituir qualquer argumento i de $\lambda\bar{w}\ F_2$ tornando-se um unificador para o par, independente da profundidade do argumento dentro do unificador parcial.

Os lemas anteriores e suas extensões levam à obtenção dos seguintes unificadores para f:

1. $\lambda\bar{w}\ F_2$ no qual não há termos $w_i[\cdots]$ como argumentos.
2. $\lambda\bar{w}\ C\ \phi(B_i \cdots w_i[\cdots h_j^k \cdots] \cdots B_s)$ com apenas um $w_i[\cdots h_j^k \cdots]$ substituindo um dos argumentos de F_2 , ($1 \leq j \leq t$) e ϕ = { outras substituições obtidas}.
3. $\lambda\bar{w}\ C\ \phi(B_i \cdots\cdots B_s)$ com apenas um $w_i[\cdots]$ substituindo recursivamente um dos argumentos de B_i e ϕ = {outras substituições obtidas}.

O universo dos unificadores de $\langle F_1, F_2 \rangle$ pode conter ainda qualquer combinação dos unificadores acima listados.

Para melhor entendimento considere-se ϕ como condicionador para a existência de um dos unificadores listados em 2 e 3 acima. Ou seja, ϕ estipula uma lista de substituições necessárias para que exista o unificador.

Como pode haver $w_i[\cdots h_i^k \cdots]$ em diversas posições, a árvore de unificação permite observar que se existirem dois unificadores com $w_i[\cdots h_i^k \cdots]$ em posições diferentes então pode-se obter um unificador que contenha os dois $w_i[\cdots h_i^k \cdots]$ desde que seja possível uma combinação entre os unificadores ϕ correspondentes à unificação dos pares $\langle A_i, B_k \rangle$. Em outras palavras, se os condicionadores puderem funcionar simultaneamente. O conceito de "unificador mais geral" está sendo usado nesta "acomodação". Dois unificadores podem ser combinados se houver um unificador mais geral que os absorva.

A proposição a seguir indica a condição de existência de um unificador combinado.

Proposição 3.1 *Sejam os unificadores entre* $\langle A_i, B_k \rangle$ *e* $\langle A_j, B_l \rangle$, $(l \neq k)$ *representados por*

$$\phi_1 = \{v_1/tv_1, v_2/tv_2 \cdots v_m/tv_m \} \text{ e } \phi_2 = \{u_1/tu_1, u_2/tu_2 \cdots u_l/tu_l\}.$$

Seja ainda definida a pseudo-composição dos dois unificadores como $\phi_1 \diamond \phi_2$, *que é a composição sujeita a não eliminação das substituições* u_i/tu_i *quando* $u_i = v_j$ *para qualquer* i *e* j. *Estes pares de substituição devem ser substituidos por uma substituição mais geral formada a partir das duas. Se isto for possível então tem-se o unificador*

$$\{\phi_2(\phi_1(f/\lambda\bar{w}\ C\ B_i \cdots w_i[\cdots h_i^k \cdots] \cdots w_j[\cdots h_i^l \cdots] \cdots B_n)) \cup \phi_1 \diamond \phi_2 \}.$$

Usa-se na proposição o conceito de que ϕ_1 e ϕ_2 são condições para que existam os unificadores $w_i[\cdots h_i \cdots]$ nas posições l e k respectivamente. Subentende-se também que a pseudo-composição é comutativa pois representa apenas a união dos dois unificadores.

A proposição indica que basta obter os unificadores parciais ϕ_1 e ϕ_2 e testar se os mesmos se "combinam".

Considere-se os ramos da árvore de unificação cujo primeiro arco é uma projeção. A intenção com estas projeções iniciais é verificar a possibilidade de se unificar pares de discórdia da forma $\langle A_i, F_2 \rangle$.

Obedecidas as condições de tipo, os primeiros unificadores parciais serão:

$$f/\lambda\bar{w}.w_i(h_1\bar{w}) \cdots (h_t\bar{w})$$

onde t tem o significado a ele atribuído anteriormente.

O primeiro conjunto discórdia resultante destas projeções é (já descrito no lema 3) constituída de t pares de discórdia da forma $\langle (h_i^k A_1 \cdots A_r), B_i \rangle$

O procedimento de obtenção dos unificadores para estas projeções segue recursivamente os conceitos estabelecidos nos lemas anteriores. No entanto, nestes casos não pode haver combinação pois a substituição para f tem uma única possibilidade para $w_i[\cdots h_i^k \cdots]$

Tem-se agora condições de visualisar um algoritmo geral para a unificação do par $\langle F_1, F_2 \rangle$.

Define-se a seguinte estrutura de árvore para a fórmula F_2: O nó raiz é rotulado por F_2. Recursivamente os nós filhos são os argumentos do nó pai. Os nós são enumerados da seguinte forma. A raiz (F_2) tem número 0 e é sub-entendida como argumento 0 (zero). Os nós subsequentes herdam a numeração do nó pai seguida do número sequencial do argumento que eles representam.

Desta forma o número 031 para o nó representa que ele é o primeiro argumento do terceiro argumento da raiz. Em cada nó, o número que o segue indica o número de abstrações daquele termo. Para cada arco, a existência de um rótulo i indica que seu nó está ligado à i-ésima abstração, sequencialmente até aquele arco da árvore.

Considere dois valores i e j, onde i representa o índice de um argumento A_i e j o número de um nó qualquer da estrutura de F_2. São unificadores para o par $\langle F_1, F_2 \rangle$, gerados por imitação ou projeção, para $(m \geq 0)$

1. Para $1 \leq i \leq r$ e $\forall\, j$ ($j \in$ {nós da árvore F_2 }) os unificadores ij são
   $\{\phi(f/\lambda \bar{w}\ C\ B_1 \cdots w_i[\cdots h_i^j \cdots] \cdots B_s) \cup unif_m(\langle A_i,$ termo do nó j $\rangle)\ \}$
   O índice j representa qual termo da fórmula F_2 deve ser substituido por $w_i[\cdots h_i^j \cdots]$ e ϕ representa $unif_m$. Se i = 0, então $f/\lambda \bar{w}\ C B_1 \cdots B_s$ é o unificador ij.

2. Para $1 \leq i \leq r$ e j=0, o unificador ij é
   $\{\phi(f/\lambda \bar{w}.w_i(h_1^k A_1 \cdots A_r) \cdots (h_t^k A_1 \cdots A_r)) \cup unif_m(\langle A_i, F_2 \rangle\ \}$
   em que t é o número de variáveis ligadas em A_i e ϕ representa $unif_m(\langle A_i, F_2 \rangle)$

Figura 2: Algoritmo ADF para obtenção dos unificadores "principais", que são o resultado de imitações e de i ($0 \leq i \leq 1$) projeções.

A Fig. 2 contém o algoritmo que obtém os unificadores nos quais apenas um arco foi obtido por meio de projeção.Ele utiliza a estrutura de árvore proposta e os obtém por meio da unificação dos argumentos de F_1 com os nodos da árvore de F_2.

Nas duas condições do algoritmo da figura 2, a união com $unif$ resulta numa nova aplicação do algoritmo, como será visto adiante. Não existirá o unificador ij se não houver unificação do par sujeito a $unif$. Este algoritmo é obtido pela aplicação direta do lema 3.

4 Combinação de Unificadores

A idéia sobre a qual se elabora o conceito de combinação de unificadores parte da seguinte observação que já foi exposta na propriedade 2. Sejam

$$\phi_1 = g/\lambda w_1 w_2\ C\ w_1\ 2 \text{ e } \phi_2 = g/\lambda w_1 w_2\ C\ 3\ w_2$$

dois unificadores para um par de discórdia. Neste caso $g/\lambda w_1 w_2 C\ w_1\ w_2$ também é um unificador para o par. Intuitivamente percebe-se que o primeiro argumento de F_2 é 3 e o segundo é 2.

Em um grande número de casos o unificador é um conjunto de substituições com as variáveis livres do par de discórdia à esquerda da substituição. Dois unificadores, para se combinarem precisam ter os unificadores de suas variáveis livres combinados.

O algoritmo COMB (Fig. 3) é usado para testar se duas substituições se combinam gerando uma terceira. Nota-se que se em ξ_1 não houver $w_i[\cdots h_i^k \cdots]$ não há combinação.

Para se combinar dois unificadores considerando-os como conjuntos de substituições para as variáveis livres de F_1 e F_2 observa-se se as substituições com a mesma variável livre à esquerda se combinam. O algoritmo COMBT (Fig. 4) foi proposto com esta finalidade .

Obtidos os unificadores principais, os demais unificadores do par de discórdia são todas

Sejam ξ_1 e ξ_2 duas substituições par uma variável livre g.

Em ξ_1 localize as posições onde há $w_i[\cdots h_i^h \cdots]$ na substituição para g (exceto no conjunto de variáveis de ligação). Use para isso a estrutura de árvore, por exemplo.

1. COMB := $\emptyset$.
2. Copie ξ_2 para ξ_3.
3. Para cada $w_i[\cdots h_i^h \cdots]$ existente no nó ijkl... da árvore da substituição para g de ξ_1

   3.1 Verifique se os nós anteriores do ramo (formado pela sequência de nós i, ij, ijk, ...($i \geq 0$) de ξ_3) não estão ocupados por algum $w_i[\cdots h_i^h \cdots]$, $(1 \leq i \leq r)$.

   3.2 Se estiver, a combinação fracassa. COMB := $\emptyset$. Vá para 4. Caso contrário acrescente $w_i[\cdots h_i^h \cdots]$ na posição ijkl... da correspondente substituição de ξ_3. COMB := ξ_3.
4. Fim {COMB é a combinação entre ξ_1 e ξ_2.}

Figura 3: Algoritmo COMB para cobinação de duas substituições para a mesma variável livre gerando um unificador mais geral.

Sejam ϕ_1 e ϕ_2 unificadores ou conjuntos de substituições.

1. COMBT := $\emptyset$.
2. $\phi_3 := \emptyset$.
3. Para cada substituição $\xi = y/t_{\phi_i}$, y pertencente às variáveis livres, $\xi \in \phi_i$, $(1 \leq i \leq 2)$,
   (a) Se não existir uma substituição y/t_{ϕ_j} em $\phi_j, j \neq i$, $\phi_3 := \phi_3 \cup \{y/t_{\phi_i}\}$.
   (b) Se $t_{\phi_1} = t_{\phi_2}$ então $\phi_3 := \phi_3 \cup \{y/t_{\phi_1}\}$.
   (c) Senão se $t_{\phi_1} \neq t_{\phi_2}$ e $COMB(\xi_{\phi_1}, \xi_{\phi_2}) = \emptyset$ então $COMBT(\phi_1, \phi_2) := \emptyset$. Va para Fim.
   Caso contrário $\phi_3 := \phi_3 \cup \{COMB(\xi_{\phi_1}, \xi_{\phi_2})\}$.
4. Fim.

Figura 4: Algoritmo COMBT para combinação de dois unificadores para um par de discórdia de $\langle F_1, F_2 \rangle$.

as combinações entre os unificadores eles. Define-se na figura 5 o algoritmo UNIF($\langle F_1, F_2 \rangle$) que obtém todos os unificadores para o par

A consequência final da aplicação do algoritmo UNIF, após a aplicação de ADF é obter um conjunto de unificadores para o par. Mostrando-se que: (1) todos os unificadores finais θ

Seja P(ADF) o conjunto potência de ADF($\langle F_1, F_2\rangle$).

Para cada elemento de P(ADF), combine seus elementos da seguinte maneira. Seja um elemento γ de P(ADF) = $\{\theta_1 \cdots \theta_n\}$

1. n = 0 então UNIF($\langle F_1, F_2\rangle$:= UNIF($\langle F_1, F_2\rangle \cup \gamma$.
2. Se n = 1 então UNIF($\langle F_1, F_2\rangle$) := UNIF($\langle F_1, F_2\rangle$) $\cup$ θ_1
3. Se n = 2 então UNIF($\langle F_1, F_2\rangle$) := UNIF($\langle F_1, F_2\rangle$) $\cup$ $COMBT(\theta_1, \theta_2)$
4. Se n$\geq$ 3 então UNIF($\langle F_1, F_2\rangle$) := UNIF($\langle F_1, F_2\rangle$) $\cup$ CC onde CC é obtido por
   CC := $\emptyset$,
   Para i = 1 até n CC :=$COMBT(CC, \theta_i)$.

Figura 5: Algoritmo de Unificação geral.

da árvore de unificação que contêm apenas um argumento da forma $w_i[\cdots hi^k \cdots]$ são gerados pelo algoritmo ADF e são os unificadores chamados de "principais", e que (2) os demais são combinações daqueles obtidos em 1, sendo a combinação definida pelo algoritmo COMBT anteriormente definido, chega-se a conclusão de que UNIF gera os mesmos unificadores que MATCH.

5 Aplicações dos Algoritmos

Utiliza-se a seguir um exemplo tirado de [3] para estabelecer, ainda que sobre alguns aspectos apenas, um contraste entre a utilização da combinação dos algoritmos MATCH e SIMPL e o algoritmo UNIF, aqui proposto. Naquele exemplo tinha-se o seguinte par $\langle F_1, F_2\rangle$, rígido-rígido

$\{$((mapfun F_1 (cons X L_1) (cons (F_1 X) L_2)),
(mapfun F_2 (cons 1 (cons 2 nil))
(cons (g 1 1) (cons (g 1 2) nil))))$\}$

que por meio da rotina SIMPL gerou o seguinte conjunto discórdia:

$\{\langle F_1, F_2\rangle, \langle X, 1\rangle, \langle L_1, (cons\ 2\ nil)\rangle, \langle (F_1\ X), (g\ 1\ 1)\rangle, \langle L_2, (cons\ (g\ 1\ 2)\ nil)\rangle\ \}$

Os pares flexível-rígido do conjunto são:

$\langle X, 1\rangle$, $\langle L_1, (cons\ 2\ nil)\rangle$, $\langle (F_1\ X), (g\ 1\ 1)\rangle$ e $\langle L_2, (cons\ (g\ 1\ 2)\ nil)\rangle$.

Apenas o terceiro par não tem ADF unitário. Os demais não têm combinações e UNIF = ADF com os unificadores

$X/1$, $L_1/(cons\ 2\ nil)$ e $L_2/(cons\ (g\ 1\ 2)\ nil)$

respectivamente. Para se encontrar estes unificadores não houve qualquer criação de termos. Apenas verificou-se a adequação de tipo. Geração de variáveis h_i não foi necessária, tampouco β-reduções.

O terceiro par foi alterado para $\langle (F_1\ 1), (g\ 1\ 1)\rangle$. Nele é gritante a vantagem de UNIF pois bastou a comparação de $\langle 1, 1\rangle$ e posteriormente $\langle 1, 1\rangle$ para se obter os três unificadores principais que são

$$\{\ F_1/\lambda w_1(g\ 1\ 1),\ F_1/\lambda w_1(g\ 1\ w_1),\ F_1/\lambda w_1(g\ w_1\ 1)\}$$

e a combinação gerou o quarto unificador que é $\{\ F_1/\lambda w_1(g\ w_1\ w_1)$.

Apesar de não representar uma regra geral, esta unificação não usa β-reduções, geração de novas variáveis livres, consulta a tipos para geração de variáveis, controle de estrutura de árvore para posterior composição dos unificadores parciais, e outros mecanismos necessários para a geração do unificador.

A unificação é feita apenas para os unificadores chamados de "principais". Os demais são conseguidos por meio de mecanismos de combinação e dispensam toda a mecânica de pesquisa incluida em MATCH. O número máximo de argumentos principais é função do número de argumentos de F^2, em qualquer profundidade, e pode ser melhor visto pela estrutura de árvore proposta. Em muitos casos ele não atinge a metade do número total de unificadores.

Como um exemplo, a unificação do par $\langle f\ 1\ (T\ 2)\ (y\ 2), C\ 1\ (P\ 1)\ (y\ 2)\rangle$ produs 12 unificadores principais. O número total de unificadores está em torno de 50, dependendo da forma como se processam os termos flexível-flexível.

6 Conclusões

Os exemplos precedentes representam um sinal de que o algoritmo UNIF usado para a unificação de pares de discórdia deve ser analisado como uma alternativa para MATCH. Ele aponta para uma melhora significativa de desempenho. Embora testes exaustivos devem ainda ser feitos antes de quaisquer conclusões, nossas observações preliminares mostram que existem vantagens que devem ser exploradas.

Além da redução representada pela diminuição do número de unificadores pesquisados, o próprio tabalho de busca destes unificadores é consideravelmente reduzido no algoritmo UNIF. Este fato pode ser observado porque

- β-reduções são quase desnecessárias.
- Geração de novas variáveis livres só ocorrem quando há abstração de algum argumento de F^1.
- Não existe repetição de segmentos na árvore de unificação.
- A rotina SIMPL é simplificada

Restritas a observações empíricas preliminares, estas vantagens conferem ao algoritmo a característica promissora indicada no inicio desta seção. Recomenda-se no entanto uma formalização teórica mais profunda que conduza a propriedades como consistência e completude.

O aprofundamento da análise deste método de unificação sinaliza que mesmo a grande preocupação com tipos existente em MATCH poderia ser significativamente reduzida. A

confirmação desta conjectura pode representar um passo a mais sobre um dos parâmetros que influenciam negativamente sobre o desempenho dos interpretadores.

7 Bibliografia

[1] G. P. Huet, "A Unification Algorithm for Typed Lambda-Calculus," *Theoretical Computer Science*, vol 1, pp 27-57, 1975.

[2] A. Ludwig, "Um Interpretador para o λ-Prolog", Tese de Doutorado, Universidade Estadual de Campinas, SP, *(to appear)*.

[3] G. Nadathur, *A Higher Order Logic as The Basis for Logic Programing.* PhD Dissertation, University of Pennsylvania, 1987.

[4] G. Nadathur and D. Miller, "Higher-Order Horn Clause," *JACM*, 37:4, pp. 777-814, 1990.

Una Alternativa Eficiente para Evaluación de Restricciones de Integridad en Bases de Datos Lógicas

Alonso Márquez

Leonid Tineo

Departamento de Computación,

Universidad Simón Bolívar,

Apartado Postal 89000, Caracas 1080A, VENEZUELA

Resumen

El garantizar que un conjunto de restricciones sea satisfecho por una Base de Datos Lógica en todo momento obliga a que cada vez que se hagan Modificaciones, deba chequearse que ésta satisfaga las restricciones. La solución debe minimizar el cómputo aprovechando el supuesto de integridad (la base de datos ante de cualquier modificación está en un estado consistente con las restricciones), y evaluando sólo las restricciones que son relevantes a la modificación. Nuestra solución se basa en la consideración del conjunto de Restricciones de Integridad como la definición de un predicado especial "contradicción" y la generación incremental del modelo de la base de datos. Para lo cual definimos una transformación sintáctica de la Base de Datos Lógica, llamada la Completación Horn, que elimina el uso del Operador de Negación implícita. Esta transformación aplica a Bases de Datos Lógicas jerarquizada, lo cual resuelve la mayoría de los casos prácticos.

Palabras Clave: bases de datos lógicas, restricciones de integridad, evaluación incremental, completacion Horn, modelo perfecto, negación, modificación

1 INTRODUCCION

La semántica más ampliamente aceptada actualmente para Bases de Datos Lógicas es la semántica de Modelo Perfecto [NICOLAS78] [CHANDRA82] . Esta coincide con la semántica de Punto Fijo [APT82] [CHANDRA82], para el caso de Bases de Datos Lógicas sin Negación.

Como es sabido, el operador de Punto Fijo es un operador monótono [LLOYD87], luego su evaluación puede hacerse de manera incremental. Así, la Evaluación Incremental provee una forma sencilla y eficiente para generar el modelo de Bases de Datos Lógicas sin Negación con Modificaciones de tipo Inserción.

En el caso de Bases de Datos Lógicas con Negación, el mecanismo de generación del modelo que se utiliza es el Punto Fijo Estratificado [CHANDRA82] , caso particular de los mecanismos de prueba por encadenamiento hacia adelante, que es no monótono. Hasta ahora no se conoce ningún mecanismo de Evaluación Incremental de este operador.

Una extensión que se ha hecho a las Bases de Datos Lógicas es la posibilidad de establecer Restricciones de Integridad [LLOYD87], esto es: un conjunto de reglas lógicas que sirven para restringir el conjunto de modelos válidos para la base de datos. No se conoce una manera eficiente de hacer el chequeo de la satisfacción de las Restricciones de Integridad y asegurar que se generen modelos permitidos.

En este trabajo proponemos una solución eficiente al problema de garantizar la satisfacción de las Restricciones de Integridad por el Modelo Perfecto de Bases de Datos Lógicas. Esta solución se basa en la Evaluación Incremental de Bases de Datos Lógicas con Negación. Mediante una transformación denominada Completación Horn, basada en la completación [CLARK78], que consiste básicamente en reemplazar el uso del operador de Negación por un manejo explícito de la Negación.

Asumimos que el lector está familiarizado con los conceptos básicos del área de Bases de Datos Lógicas [GALLAIRE84] [ULLMAN88] [CERI89].

2 COMPLETACION HORN

La conocida Teoría de Pruebas nos da una manera sencilla de inferir las nuevas consecuencias que forman parte del modelo de la Base de Datos Lógica resultante de una operación de Modificación, esto debido a que se tienen reglas que definen a los predicados por comprensión.

En la Base de Datos Lógica no hay manera, de acuerdo a la Teoría de Pruebas, de inferir cuáles son las consecuencias que forman parte del modelo de la Base de Datos Lógica antes de una Modificación y no forman parte del modelo de la Base de Datos Lógica resultante de la Modificación. En consecuencia, si se está aplicando una estrategias de encadenamiento hacia adelante, habría que calcular en cada modificación todo el modelo de la Base de Datos Lógica.

2.1 Definición de la Completación Horn.

La Completación Horn de un Programa Lógico se basa principalmente en eliminar el uso del Operador de Negación y definir de manera explícita la Negación de los predicados del Programa Lógico. Para ello se hace uso de la completación de programas.

Se define la Eliminación de la Negación de una fórmula F (elimneg(F)) como:

- ri(t1,..,tai) si F= ri(t1,..,tai)
- no_ri(t1,..,tai) si F= not ri(t1,..,tai)
- elimneg(A) $\wedge$ elimneg(B) si F= (A$\wedge$B)
- elimneg(A) v elimneg(B) si F= (AvB)
- elimneg(A) $\leftarrow$ elimneg(B) si F= (A$\leftarrow$B)
- $\forall$X elimneg(A) si F= $\forall$X A
- $\exists$X elimneg(A) si F= $\exists$X A

- elimneg(not A) v elimneg(not B) si F= not(A∧B)
- elimneg(not A)∧elimneg(not B) si F= not(A∨B)
- ∃X elimneg(not A) si F= not ∀X A
- ∀X elimneg(not A) si F= not ∃X A

La Completación Horn de un predicado r (comp_horn(r)) es:

- Si r es un predicado extensional

  {r(t1,..,ta) / r(t1,..,ta) ∈ BDE} U

  {no_r(t1,..,ta) / r(ti,..,ta) ∉ BDE}

- Si r es un predicado intensional y la completación Clark de r es

  ∀X1.. ∀Xa (ri(X1,..,Xa) ↔ F)

  {elimneg(∀X1.. ∀Xa (ri(X1,..,Xai) ← F)),

  elimneg(∀X1.. ∀Xa (not ri(X1,..,Xai) ← not F))

  }

La Completación Horn del programa PL (comp_horn(PL)) es:

- comp_horn(PL) = { R / R ∈ comp_horn(r), r predicado }

Por ejemplo, consideremos la Base de Datos Lógica:

BDL = { usa(X1,X2) :- X2=sparc, dicta(X1,bases_de_datos)., dicta(esther,bases_de_datos).}

La Completación Horn es:

comp_horn(BDL) = { ∀X1 ∀X2 (usa(X1,X2) ← X2=sparc, dicta(X1,bases_de_datos)).,
∀X1 ∀X2 (no_usa(X1,X2) ← X2≠ sparc; no_dicta(X1,bases_de_datos))., dicta(esther,bases_de_datos).
no_dicta(esther,esther)., no_dicta(bases_de_datos,bases_de_datos)., no_dicta(bases_de_datos,esther).}

2.2 Justificación de la Completación Horn.

Debemos asegurar que el Modelo Mínimo de la Base de Datos Lógica resultante de la Completación Horn restringida a los predicados del programa original coincida con el Modelo Perfecto de la Base de Datos Lógica original y que la parte correspondiente a los predicados "negativos" coincida con el complemento del Modelo preferencial del programa original con respecto a la Base de Herbrand del lenguaje del programa original.

Completitud de la Completación Horn: si BDL es una Base de Datos Lógica jerárquica y F es una fórmula, entonces comp_horn(BDL) $\models$ elimneg(F) si y sólo si comp_horn(BDL) $\not\models$ elimneg(not F).

Teorema Fundamental: si BDL es una Base de Datos Lógica jerárquica y F es una fórmula, entonces BDL $\models$ F si y sólo si comp_horn(BDL) $\models$ elimneg(F).

Estos resultados se demuestran por inducción sobre la jerarquización de los predicados de la Base de Datos Lógica e inducción sobre la estructura de la fórmula F. No presentamos estas demostraciones aquí por razones de espacio, la demostración formal puede conseguirse en [TINEO92].

3 EVALUACION INCREMENTAL DE BASES DE DATOS LOGICAS.

En general, si U es una operación de Modificación, al aplicarla sobre una Base de Datos Lógica, el resultado es otra Base de Datos Lógica cuyo Modelo Perfecto viene dado por el Modelo Perfecto de la Base de Datos Lógica original más las consecuencias nuevas que pueden alcanzarse en la nueva Base de Datos Lógica menos las consecuencias que eran alcanzables en la Base de Datos Lógica original pero no lo son en la nueva [MACHANDA88] [ULLMAN88].

Por la propiedad de monotonicidad del operador de Punto Fijo se obtiene que el modelo

resultante de la Modificación en el caso de inserciones en bases de datos sin negación, es el Punto Fijo (PF) del operador de consecuencia directa (cd) aplicado al modelo previo a la Modificación con el nuevo hecho.

Mediante la Completación Horn hemos dado una definición explícita de la Negación; más aún, el Teorema Fundamental garantiza que la semántica de la Completación Horn es equivalente (vía la Eliminación de la Negación "elimneg") a la semántica del Modelo Perfecto.

De esta manera la nueva parte del modelo que se infiere ante una Inserción para los predicados "negativos" de la Completación Horn corresponden a la parte del modelo que deja de ser consecuencia luego de la actualización en al Base de Datos Lógica.

El problema de propagar las consecuencias de la Eliminación es análogo al hacer Inserciones en una Base de Datos Lógica con Negación, es decir se quiere inferir la parte positiva (nuevas consecuencias) así como la negativa (consecuencias que dejan de poder alcanzarse).

Surge una vez más como solución el uso de la Completación Horn, con la cual se le da una definición explícita a las consecuencias negativas. En este caso Eliminar A es equivalente a Insertar el átomo correspondiente a la negación de A en la Completación Horn.

4 EVALUACION DE LAS RESTRICCIONES DE INTEGRIDAD

Proponemos extender la base de datos intensional con un nuevo predicado, al cual llamaremos "contradicción", cuya definición viene dada por un conjunto de reglas, cuyos cuerpos son los cuerpos de las cláusulas Horn negativas que especifican las Restricciones de Integridad. Bajo la citada transformación, la base de datos es consistente si y solo si "contradicción" no es parte del modelo.

Dado que "contradicción" es un nuevo predicado de la Base de Datos Intensional, podemos conseguir siempre una Jerarquización donde el predicado contradicción se encuentra en el último nivel, de esta manera, evaluamos el predicado contradicción, una vez que se ha

generado el nuevo modelo de la base de datos.

Ahora bien, una vez hecha la generación incremental del modelo, si se viola una Restricción de Integridad, la base de datos ha quedado en un estado inconsistente. Hay que restaurar la consistencia de la base de datos, la solución es deshacer la Modificación realizada. Esto se logra aplicando la Modificación inversa.

5 CONCLUSIONES

Como la Completación Horn da una definición explícita de la Negación de los predicados, al aplicar operaciones de Modificación es posible deducir no sólo el conjunto de consecuencias que se agregan, sino, también el conjunto de consecuencias que eran alcanzables en la Base de Datos Lógica previa a la Modificación y que no lo son luego.

La solución que hemos propuesto para la generación de modelo de Bases de Datos Lógicas con Negación es mucho más eficiente que la solución que deberíamos hacer comúnmente de genera todo el modelo cada vez que hay una Modificación, pues aprovecha el modelo calculado previamente haciendo uso del operador de Punto Fijo que sólo hará los cálculos necesarios para obtener las nuevas consecuencias.

Para el manejo de Restricciones de Integridad definimos la Base de Datos Lógica Aumentada con el predicado "contradicción" como la Base de Datos Lógica donde se ha creado un nuevo predicado, llamado "contradicción", cuyas reglas tienen como cuerpo cada una de las Restricciones de Integridad.

Para garantizar que la Base de Datos Lógica es consistente en todo momento, rechazamos las modificaciones que hagan que "contradicción" sea parte del modelo. Debido a las propiedades de la Completación Horn, el rechazar una Modificación es equivalente a aplicar la operación inversa sobre resultado de dicha Modificación.

Nuestro trabajo fue centrado en el problema de evaluación de Bases de Datos Lógicas Jerarquizadas, principalmente teniendo en cuenta que este es un caso interesante, de com-

plejidad no elevada y que resuelve la mayoría de los problema prácticos; sin embargo, puede ser extendido a Bases de Datos Lógicas Estratificadas.

6 REFERENCIAS

[APT82] Apt, K. R. and M. H. Van Emden, "Contributions to the theory of logic programming", J. ACM 1982, 29:3.

[CERI89] Ceri, S., G. Gottlob and L. Tanca, "What you always wanted to know about Datalog (and never dared to ask)", IEEE Transactions on knowdledge and data engineering, 1:1, 1989.

[CHANDRA82] Chandra, A. K. and D. Harel, "Structure and complexity of relational queries", J. Computer and System Sciences 1982, 25:1.

[CLARK78] Clark, K. L., "Negation as Failure", in Logic and Data Bases, Gallaire, H. and J. Minker (eds), Plenum Press, New York, 1978.

[GALLAIRE84] Gallaire, H., J. Minker and J. M. Nicolas, "Logic and Databases: A Deductive Approach", Computing Surveys 1984, 16:2.

[LLOYD87] Lloyd, J. W., Foundations of Logic Programming. Springer-Verlag, Berlin, 1987.

[NICOLAS78] Nicolas, J. M. and H. Gallaire, "Data Base: Theory vs Interpretation", in Logic and Data Bases, Gallaire, H. and J. Minker (eds), Plenum Press, New York, 1978.

[SADRI88] Sadri, F. and R. Kowalski, "A theorem-proving approach to database integrity", in Foundations of Deductive Databases and Logic Programming, Minker, J. (ed) Morgan Kaufmann Publisher, 1988.

[TINEO92] Tineo, L., "Evaluación incremental de restricciones de integridad en bases de datos logicas", MSc. thesis, Universidad Simón Bolívar, 1992.

[ULLMAN88] Ullman, J. D., Principles of Database and Knowledge Base Systems Vollume I, Computer Science Press, 1988.

Towards The Automatic Validation of Conceptual Specifications For Database Systems - A Formal Approach

Alfio Ricardo Martini Jose M. V. de Castilho, Phd

Instituto de Informatica e PGCC da UFRGS - RS - Brazil
Fax : (051)3365576
E-Mail : MARTINI@INF.UFRGS.BR

Abstract

This paper describes a software tool for the rapid prototyping of database systems, based on their formal conceptual specifications in logic. The prototypes are generated as PROLOG programs. This approach enables one to get user's hands on experience with the software system before its implementation. Throughout this paper, we discuss basic reasons for such formal approach, the structure of the specification language, as well as an overview of the software tool architecture. At the end, we present some conclusions about the contribution of this work to enhance end-users satisfaction with the final product of the data base design procedure.

Key words: first order logic, database systems, conceptual design, formal specifications.

1 Introduction

The conceptual design for database systems, sometimes called data modeling, involves formulating the data objects of the universe of discourse in terms of some data modeling formalism to produce a specification of what it is that the required database is to represent [13].

Informal specifications alone are certainly not appropriate, because they are incomplete, sometimes inconsistent, inaccurate and ambiguous, and they rapidly become bulky, unstructured and therefore non reliable.

Nowadays, in the software engineering community, there is a general agreement that in large or complex software developments (e.g., database systems), formal specifications are a must in order to obtain high quality software [7].

As a consequence, among many others, obvious advantages arise when a formal approach for developing complex software objects is taken [8]:

- They play a role of contract between users and implementor;
- They are a starting point for automatic verification and validation of software systems;
- They can be directly (or, at least, transformed to a suitable form before being) executed;

Rapid (computer-aided) prototyping provides a means for building the scaled-down version of a system more quickly than using other conventional approaches. The final product of the prototyping activity is a working model that can be used for many purposes, such as requirements validation, feasibility study for the full system, and functional specification of a system design.

This work proposes a strategy for systematic construction of system prototypes, from database conceptual models which are formal (rather than informal, such as, for instance, the E-R Model [2]).

In the following sections we discuss the structure of the formal language, an overview of a software tool for the rapid prototyping of database systems and its basic functionality. At the end we present some conclusions about the relevance of this work.

2 The Specification Language

Logic [4] is the most powerful (and general) formal description language known, although not quite user-friendly. As a result, it is reasonable to suppose that it could be the basis of a good communication language between a user and an automatic programming system [10]. Also, besides having a solid theoretical basis upon which one can pursue database theory in general (as discussed in [6]), logic has a *constructive* characteristic, i.e., it enables rapid prototyping by compilation to a executable lower level language.

The conceptual models considered in this work are presented as formal specifications expressed in a mathematical logic language which enables one to describe database systems in a very high level of abstraction [1].

Like any other one-sorted first-order language, the present specification language alphabet consists of primitive symbols such as (1) parenthesis, (2) variables, (3) the

usual connectives, ~ (not), ^ (and), V (or), => (implication), <=> (equivalence), and (4) ForAll and Exists, for the universal and existential quantifiers, respectively.

Furthermore, the language has precisely four 2-place predicate symbols used in infix notation; *_compose_*, *_spec_*, *_type_*, and *_elem_*.

These predicate symbols are based in the "entity type constructor" concept, proposed in the E Model ([11], [12]), a semantic data model for database modeling, which is basically a data type extension to the Entity-Relationship Model [2]. Intuitively, *_compose_* relates entity instances to (each one) of their component entity instances: *_spec_* relates entity instances to their specialized entity instances; *_elem_* relates entity instances of set type to their element instances; and *_type_* relates entity instances to their type denotations.

There are also two 1-place function symbols; *rep(_)* and *card(_)*. The first delivers an integer (or a string) representation of the value of its entity instance argument, and the second, the cardinality of an entity instances set. Constants, denoted by character strings between quotes or by integer numbers, are also part of the language alphabet.

Terms, or arguments, for the predicate and function symbols are entity variables, except for "type", where the second argument is an entity type denotation

A formal specification, in this approach, consists of three parts:

- The declaration of constructed or primitive Entity Types (the "structural" part of the specification) of Integrity Constraints
- The declaration of Update Operations (specified through their pre and post conditions expressed in the logic language),and
- The declaration of Query Operations (also expressed in logic);

As an example, parts of a specification are shown below:

```
SPECIFICATION Employment_Agency

   STRUCTURE:
      person := STRING
      company := STRING
      project := STRING
      contract = person X company
      task = person X project
      .
      .
      .
   CONSTRAINTS:
```

```
    FORALL P, F
       ((P type person ^ F type task
         ^ P compose F) => EXISTS C
        (C type contract ^ P compose C))
    .
    .
    .
OPERATIONS:

    apply(p:string):
    (PRE: not EXISTS P
            (P type person ^ rep(P) = p)
     POST: EXISTS P
             (P type person ^ rep(P) = p))
    .
    .
    .
END_SPEC
```

3 Prototyping System's Overview

The system is basically a translator from the formal specification to PROLOG [3] Horn clauses. Its architecture can be easily understood by examining Fig 1.

The syntactic/semantic analyzer was written in C, and is responsible for checking the well-formedness of the specification as well as its desirable semantic properties (correctness of variable and type declarations, symbol table manipulation, etc...). As an output from this module, one gets a file containing all the integrity constraints and update/query operations expressed in a suitable intermediate language, which can be input to the translator itself.

The translator module is written in C-PROLOG, and consists of four other modules. The *data structure translator* module builds from the data structure definition part of the system's specification, a set of logic formulas that explain in full the meaning of the shorthand notation text defining the database structure.

The *integrity constraints translator* module constructs from logic formulas expressing integrity conditions imposed on the data of the system's database, PROLOG commands (rules) that could perform the corresponding integrity checking. This module is also used to translate, into PROLOG commands, the *structural* formulas produced by the *data structure module.* Both sets of PROLOG commands are used later, during the system's prototyping phase, to test the quality of the system's database contents.

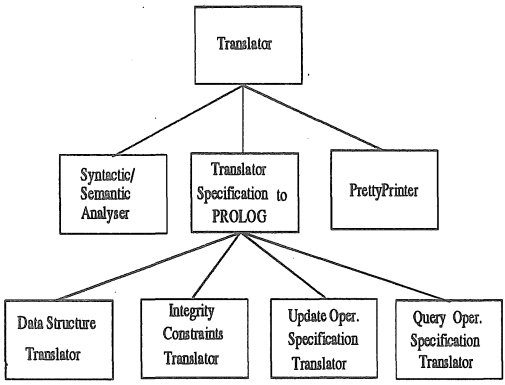


Figure 1: Program's main Architecture

The next module, the *update operations specification translator*, detailed in Fig. 2, executes the main part of the translation process. It synthesizes, from the update operations specification part of the full systems's specification, a set of PROLOG predicates that should behave, when called, like the specified operations of the database system. They consider the existence of a base of "facts" which plays the role of the system's database. As it comprehends the most characteristic and creative part of the translator, this module's behavior will be explained through a small example in the next section.

The last module performs the translation of the query operations specifications to PROLOG queries. It is practically identical to the integrity constraints translator.

4 Update Operations Synthesys

The update operations synthesis profits from the fact that both the specification language and PROLOG are offsprings from first order logic languages. So, some translation steps seem to be rather trivial. However, PROLOG has some procedural characteristics that had to be taken care of in many other steps, so as to accomplish the desired translation.

The module is composed by three submodules : *normal form transformation, atomic formulas translator, and PROLOG operation composition.* The first one implements a series of simbolic transformations that produces from the original pre and post- conditions update operations specification, a formula in prenex normal form, prefixed with only existential quantifiers, negated or not, and whose body is a disjunction of conjuctions of atomic formulas, negated or not. The presentation of the synthesis example starts at this point. Let the normalized formula, representing an

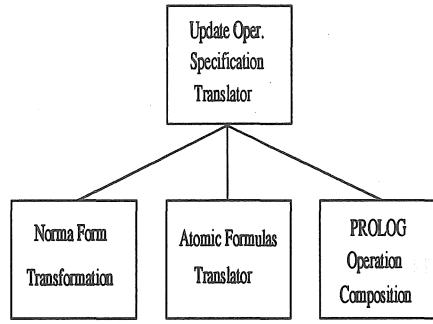


Figure 2: Update Operations Translator Architecture

aplly for a *job* operation (the database system considered here is more or less the same presented in [5]), where a person not already present in the database apllies as candidate for a job (see the apply specification in the example presented in section 2).

```
i)      ~There_Exists Pa There_Exists Pb
            (Pa type person ^ rep(Pa) = p
             ^ Pb m-type person ^ m-rep(Pb) = p)
```

Pa e Pb are variables that may assume entity instances as values, p is (the name of) the person that is applying as candidate for a job; Pa *type* person is an atomic formula that is true if the entity instance denoted by Pa is of type "person". The function symbol *rep*(Pa) yields (the name of) the person denoted by p. The prefix *m-* marks the corresponding atomic formulas as post condition atomic formulas.

In the atomic formulas translation process, each atomic formula is translated to some PROLOG construct, according to the kind of quantification to which it is submitted, and to its condition as negated or not. A small knowledge base of pairs $< atomic_formula_pattern, PROLOG_construct_pattern >$ is used to perform each atomic formula translation. In the example, the following sequence of constructs is obtained:

```
ii)         not(type(Pa,person), r(Pa,p)),
            gs(Pb), assert(type(Pb,person)),
            assert(r(Pb,p))
```

After this translation step, the following task is to build, as a PROLOG predicate, the whole operation.

```
iii)        operation(apply,p) :-
                    not(type(Pa,person),  r(Pa,p)),
                    gs(Pb), assert(type(Pb,person)),
                    assert(r(Pb,p)), !.

            operation(apply,p) :- put("pre-condition failure"), true.
```

5 Conclusions

In this paper a software tool was described for the automatic generation of databases systems prototypes from their formal conceptual models. A first order language was addopted because of its power, general expressiveness, and its "constructive approach".

Our approach of generating database systems prototypes from their formal conceptual models has interesting advantages:

- The prototypes can be generated very quickly, enabling designers and users to test a series of different system's functionalities.
- Users can make validating experiments with the prototypes and thus, specified requirements can be easily and efficiently revised and improved.
- The formal nature of the specification provides a basis for the use of mathematical reasoning, for the automatic verification of correctness of databases specifications.

References

[1] CASTILHO, J. M. V. de *Especificacoes Formais em Logica para o Projeto de Bancos de Dados*, Tese de Doutorado, DI-PUC/RJ, 1982.

[2] CHEN, P. P. The Entity-Relationship Model-Toward a Unified View of Data, *ACM Transactions on Database Systems*, 1(1), pp.9-36 (March 1976).

[3] CLOCKSIN, W. F. and MELLISH, C. S. *Programming in Prolog*, 2. ed., Berlin, Springer-Verlag, 1984.

[4] ENDERTON, H. B. *A Mathematical Introduction to Logic*, Academic Press, New York, 1972.

[5] VELOSO, P. A. S, CASTILHO, J. M. V. de, FURTADO, A. L. Systematic Derivation of Complementary Specifications, In:*Proceedings 7th VLDB Conference*, Cannes, pp. 409-421 (1981).

[6] GALLARE, H. and MINKER, J. Logic and Databases: A Deductive Approach, *ACM Computing Surveys*, Vol. 16, No. 2, June 1984, pp. 153-185.

[7] HOREBECK, I.V. and LEWI, J. *Algebraic Specifications in Software Engineering - An Introduction*, Springer-Verlag (1989).

[8] JONES, C. B. *Software Development - A Rigorous Approach*, Prentice-Hall International (1980).

[9] LUQI, Software Evolution Trough Rapid Prototyping, *Computer*, Vol. 22, No. 5, May-1989, pp. 15-25.

[10] RICH, C. and WATERS, C. R. Automatic Programming: Myths and Prospects, *Computer*, Vol. 21, No. 8, August 1988, pp. 40-51.

[11] SANTOS, C.S. dos, NEWHOLD, E.J., FURTADO, A.L. A Data-Type Approach to the Entity-Relationship Model, In : P. P. Chen (ed), *Entity-Relationship approach to systems analysis and design*, North-Holland, 1980.

[12] SANTOS, C. S. dos, OLIVEIRA, J. P. W. de, CASTILHO, J. M. V. de O Modelo E, In : *Anais Semish 13*, Olinda, jul 1986, pp. 342-350.

[13] WHITTINGTON, R.P. *Database Systems Engineering*, Clarendo Press, Oxford (1988).

MACOBJ - A Class-Based Tool for Numerical Problem Solving

Alfio Ricardo Martini Marcio Serolli Pinho
Tiaraju Asmuz Diverio, Msc

Instituto de Informatica e PGCC da UFRGS - RS - Brazil
Fax : (051)3365576
E-Mail : MARTINI@INF.UFRGS.BR

Abstract

The purpose of this paper is to present a tool for numerical problem solving (MACOBJ), which is based on a command language interaction style user-interface. The user-interface provides the the user with a very simple and consistent conceptual model, which is based on the object-oriented paradigm. Firstly,we introduce the notion of object-oriented user-interfaces. After, we present the classes of mathematical objects offered to the user, and an overview of the system's functionality and its symbolic capabilities.

Key words: object-oriented design, command languages, numerical calculus, conceptual model.

1 Introduction

MACOBJ represents a natural evolution from a previous design [5] which was also based on a command language interaction style.

After prototyping a series of test designs, we concluded that a very simple and coherent user's model will have to be developed so as to overcome the known problems related to command languages based user-interfaces [7].

On our last design we proposed the following requirements for the software tool :

- It should cover a *broad array* of computational methods;

- It should be *objective* (i.e., with a few steps the user should accomplish what it wanted) and *powerful* enough to be used by domain experts (e.g., teachers and researchers of the related field);
- It should present the user with a very well defined and *simple conceptual model*, so as to enable him (or her) to apply always the same semantic strategies throughout his (or her) application tasks;

The object-oriented design methodology provided us with useful concepts to create a very consistent and simple user's model, as well as to refine our command language grammar, by applying orthogonalization (factorization) [3] on some language parameters from our previous language description.

Throughout the following, we discuss basic object-oriented concepts related to user-interfaces and MACOBJ's basic functionality. We assume that the reader is familiar with the basic object-oriented terminology, and its benefits to the computer field as well ([4], [BOO91]).

2 Object-Oriented User-Interfaces

An object-oriented system can be understood as a collection of objects. Each object implements a little piece of the total system or application behaviour.

When building an object-oriented user-interface one of the first things to deal with is the global system's object organization. In order to design and implement an user-interface component , the designer needs to specify what kind of communication can occur between the objects. The possible messages which can be sent or received by an object must be clearly defined in its class interface (message protocol).

An object-based user interface enables one to create objects which are typeless (not belonging to any class). A class-based user interface permits the creation of typed objects, and in an object-oriented user-interface it is possible to create typed objects that, possibly, inherit some properties from some other classes [8] (see Fig. 1).

MACOBJ's user-interface is class-based, because it does not yet provide any mechanism for the user to express inheritance relationships between the objects created during the application task.

3 The Conceptual Model

The user's conceptual model is represented by the information handled by the user and the processes applied to this information [6]. The model enables him (or her) to

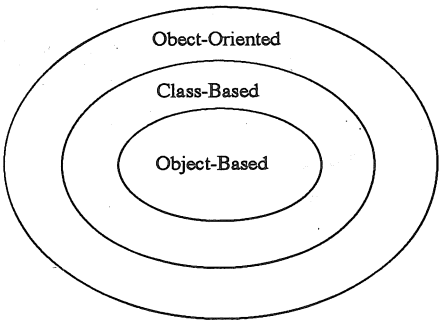


Figure 1: Object-Orientation and User-Interfaces

develop, with or without computer knowledge, a broad comprehension about what the program is doing. By understanding the conceptual model, he can anticipate the effect of his actions and develop strategies to correctly operate the program.

In the MACOBJ's conceptual model, the user creates an object by issuing a command that simultaneously define the object's name and its class. Therefore, when wanting to perform some computation on the object, the user needs just to edit the appropriate command (with its necessary parameters, if any), and it will automatically apply to the *current selected object* (*CSO*), i.e., the one that was last created. The CSO paradigm (as defined in [3]) was obtained by factoring the object parameter from our previous grammar definition. The CSO paradigm also enables that all the common operations that are applicable to objects of any class, be issued with the same syntax. The appropriate method is determined by the *current selected class*. This approach relieves the name space congestion problem. That it is to say, he , in a given state of its application task, needs just to concentrate on a smaller sets of commands, rather than worrying with all the ones available in the tool's user-interface.

4 System's Functionality.

MACOBJ's user-interface presents the user with four pre-defined mathematical classes (*Mathematical Expressions*, *Ordered-Pairs*, *Variables* and *Differential Equations*) upon which he can create objects and realize a series of computations. The E-R [2] diagram, in Fig. 2, present these classes and how they are related to each other.

The interpretation of the E-R diagram is straightforward, and it says, for instance, that an object of the class *Mathematical Expressions* can create up to n (n = 1, 2, 3,...) objects of the class *Ordered-Pairs*. Also, an object of the class *Mathematical Expressions* can use resources from up to n (here, n is implementation defined) objects

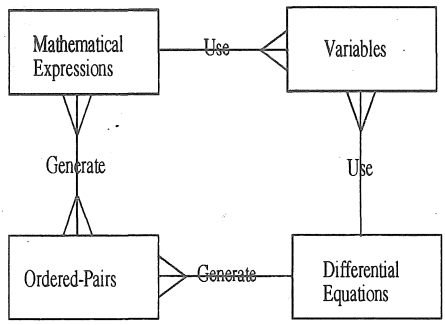


Figure 2: E-R diagram for MACOBJ's classes

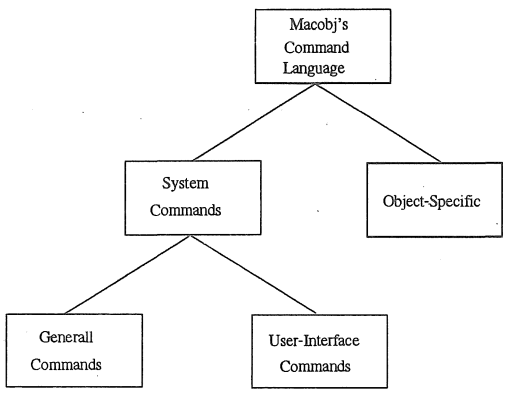


Figure 3: MACOBJ's command language hierarchy

of the class *Variables*. These relationships are supported by a series of methods offered by each class interface.

Some commands, so-called *system commands*, can be issued at any time. Other commands, *object specific commands*, can only be issued once an object has been selected. So, the range of object-specific commands is determined by the type (class) of the *CSO*, and will vary from class to class. The system commands are subdivided in *generall commands*, i.e, its behavior depends upon the current selected class, and *user-interface commands*, which are commands for object visualization and context sensitive help, basically (Fig. 3).

A major advantage of the class based style of command invocation is that it permits user's to concentrate on what it is they want to accomplish, and not how it is that this task is to be performed. Individual objects themselves are responsible for knowing the how, leaving the user to work at a much higher level.

In order to present a broad array of computational methods, MACOBJ's user-interface enables the user to perform computations of zero finding, polynomial fitting, numerical integration, derivative evaluation, and ordinary differential equations solving.

5 Conclusions

In this paper we have described a class-based tool for numerical problem solving, MACOBJ.

We used object-oriented concepts to create an suitable abstraction to develop MACOBJ's conceptual model. The conceptual model is based on the CSO paradigm, and it is enabled by applying a process called orthogonalization on the object parameter of the command language grammar. This paradigm enables the user to see the application as a consistent model, letting him develop homogeneous strategies to operate all the objects in the application regardless of its class (type). Also, the command language interaction style, together with a broad array of computational methods, represents a powerful tool for the domain expert user to quickly solve the common problems related to numerical calculus.

References

[1] BOOCH, G. *Object-Oriented Design with Applications.* The Benjamin/ Cummings Publishing Company, Inc, California, 1991.

[2] CHEN, P. P.,The Entity-Relationship Model - Toward a Unified View of Data, *ACM Transactions on Database Systems* , 1(1), pp. 9-36 (March 1976).

[3] FOLEY, J. Model and Tools for the Designers of User-Computer Interfaces. In : *Theoretical Foundations of Computer Graphics and Cad* , Italy, July 1987.

[4] MEYER, B. *Object-Oriented Software Construction* , Prentice-Hall, New-York, 1988.

[5] MARTINI, A.R. and DIVERIO, T.A. A Descricao de uma Interface Grafica e Interativa para Matematica Computacional. CONFERENCIA LATINO-AMERICANA DE INFORMATICA, XVII, 1991, 8-12 julho, Caracas,*Annales*, CLEI 1991, pp. 949-965.

[6] NEWMAN, M.W. and SPROULL, F.R. *Principles of Interactive Computer Graphics* . MacGraw-Hill, New York, 1979

[7] SHNEIDERMANN, B. *Designing the User-Interface : Strategies for Effective Human-Computer Interaction* . Addison-Wesley, Reading, 1987.

[8] WEGNER, P. Learning the Language, *Byte*, pp. 245-253, March 1989.

DYNAMIC PROPERTIES OF STATECHARTS: A REACHABILITY TREE AND ANALYSIS OF SOME PROPERTIES

P. C. Masiero
Ines G. Boaventura
J.C. Maldonado
Department of Computer Science and Statistics - ICMSC
University of São Paulo, Brazil - P. O. Box 668, 13560 - São Carlos, SP

Abstract

In this paper we present an algorithm to create a reachability tree for statecharts and show how to use this tree to analyze dynamical properties of statecharts. The properties analyzed are: reachability from any state configuration, usage of transitions, deadlocks, and valid sequence of events. Statecharts are a recent extension to finite state machines with capability for expressing hierarchical decomposition and parallelism. They have also a mechanism called history, to remember the last visit to a superstate. These features make it more difficult building a reachability tree for statecharts and we show how these problems were solved in the tree we propose.

Keywords: Statecharts, reactive systems, reachability tree, exhaustive simulation, dynamic properties of statecharts.

Acknowledgement: The first author has financial support from CNPq.

1 Introduction

Statecharts are a visual formalism for specification of reactive systems, proposed by Harel, in [4]. They are an extension to conventional state-transition diagrams based on finite state machines. Its outstanding features are hierarchy of states, parallelism and a communication mechanism via broadcasting. It also features a rich set of special notations for augmenting the notational power thus allowing the specification of complex problems in very concise diagrams. This is also the main reason why it is difficult to validate a statechart against its intended behaviour.

A possible solution to this problem is the exhaustive simulation of the statechart to produce all possible scenarios. However, the number of different possibilities may be very large, rendering impractical any exhaustive search. This problem does not have a solution but can be circumvented if we simulate selected parts of large statecharts. This is greatly facilitated by the diagram's hierarchical nature. The same problem can be attributed to Petri Nets, for which the methods proposed so far for verification of complex systems are usually based on choosing partial specifications to validate [7]. The solution we propose here is based to some extent on the reachability tree already developed to simulate the different configurations of a Petri Net [9].

The tool called STATEMATE has already implemented exhaustive search algorithms to many of the properties we will show in this paper [6], [3]. We, nevertheless, believe the solution presented here is relevant because it is based on a concept of reachability tree which is very well established. Also, after a literature review, we have not

found any other solution to this problem and the method used by STATEMATE is not public domain.

The structure of this paper is as follows. After a brief overview of statecharts, a reachability tree for statecharts is proposed in Section 2. Also in this section, one example of statechart and its corresponding tree is presented. Next, in Section 3, based on the reachability tree proposed, several dynamic properties of statecharts are defined. It is also briefly shown how they can be verified using the reachability tree.

2 Defining a Reachability Tree for Statecharts

The basic components of the statecharts are the *states*, as in the conventional state-transition diagrams. However, states may be embedded into *superstates* creating hierarchies of states. The superstates may be of two types: AND-States or XOR-States. The components of an AND-State are called *orthogonal* components and have the distinctive feature that when a system is in an AND-State it is also in all its orthogonal components. Conversely, when the system is in an XOR-State, it must be in only one of its substates.

States interact through *transitions* which have three basic parts: an event name, a condition, and action statements. These three components appear as a label attached to the transition and have the syntax: $event - expression[condition]/action$. The event expression can be null if there is a condition. The action part is also optional. A transition fires when both the event-expression and the condition evaluate to true. In this case, the actions are executed. Execution of an action may generate other events which are broadcast to the orthogonal components possibly firing new transitions. Assignment of arithmetic and logical expressions to variables is also allowed as legal action statements.

Execution of a statechart is based on a sequence of time steps where, in each step, events generated by the environment, added to the ones generated internally as a consequence of action execution, cause a set of transitions to be fired. The intended semantics is that firing a transition takes no time while being in a state takes a certain time [5]. The example in Figure 1 shows that state A is an XOR-superstate and has five substates: B, C, G, H and D. Only one of them can be active when A is active. State F has two orthogonal components: F1 and F2. When the transition labeled by event g fires, F is entered and the basic states F11 and F21 are activated explicitly; F1 and F2, as well as F, are also activated implicitly.

The history symbol attached to the transition labeled by event f means that the default state E will be activated the first time that transition fires. This will never in fact happen in this diagram because E is the default state and will be activated together with A. Afterwards E or F will be chosen, whichever was last visited. If it is the case that F is chosen, then its default states will be activated. When we have recursive history (H*) the actual basic states last visited will be activated, as in the case of the transition labeled d, leaving from H.

The reachability tree proposed is based on statechart configurations. The following notation for representing a configuration is used: parentheses to represent XOR-superstates; brackets to represent AND-superstates; and, the values 1 and 0 to represent basic states, showing whether they are active or not active, respectively. Hence,

be noticed however, that we will consider each single transition and not more than one ocurring at the same time step as is the semantics defined in [5] and [8].

For each element in this sequence we take the set $SR = \{(\Upsilon_j, \Pi_j^*)\}$ of all possible system reactions. In the tree, each node will represent a statechart configuration. They are classified into four different types: new configuration, old configuration, terminal configuration and history configuration. After being computed, a configuration will be classified as new if it was not already in the tree; otherwise, it will be classified as an old configuration. A configuration will be terminal when it does not have any enabled transition, i.e., SR=ϕ. A history configuration represents all configurations reachable from transitions which have a history symbol attached. We will use the symbols "H" and "*" to represent this case. When eventually we search the tree, the history symbols will be replaced by the last configuration of those states in the scope of the history symbol. This will be clarified later with an example.

The initial (or default) configuration is the reachability tree's root node. Considering it as a new configuration to be processed, we take the set of all transitions which will cause this configuration to be left explicitly or implicitly. For each transition taken, a new step is computed generating the next configuration which is classified and appropriatelly inserted in the tree. This procedure is then repeated for each new configuration until there is none of them. The algorithm is listed in Figure 3.

Statechart semantics also permits many events to happen in a time instant thus firing different transitions in orthogonal components. We decided not to combine the different transitions enabled in orthogonal components. This would make the algorithm much more complex because in the case of more than two transitions enabled in orthogonal components we would have to consider all possible combinations.

The reachability tree shown in Figure 2 corresponds to the statechart of Figure 1. In it we can see that when we activate a node by history we simply place "H" or "*" in the coordinates corresponding to states within the history scope. They will be replaced by the correct sub-configuration by the search algorithms as will be seen in the next section. A problem caused by history occurs when it is possible to reach a node which was never activated before. This cannot happen with the statechart of Figure 1 as all its states with history symbols associated are activated the first time the external state is activated. If instead we had another state as the default state of A, it would not be possible to have as the only arc leaving the initial configuration one leading to a history node. We create then two nodes, one for the default configuration of the states affected by history and another for the conventional history node. The former has its arcs labeled "<event expression>/def" and the latter has the usual "<event expression>" label. When we search such a reachability tree, the "/def" arc will be visited only once and discharged.

3 Dynamic Properties Verified through the Reachability Tree

Using the reachability tree proposed in Section 3 we can analyze several properties of a given statechart. See [1] for a thorough treatment.

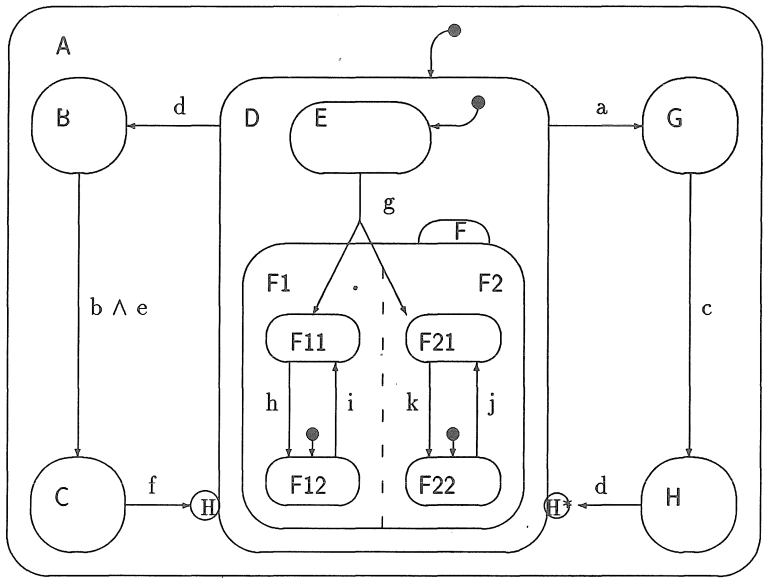


Figure 1: A Statechart with many features

the notation preserves hierarchy and decomposition type but only the basic states are explicitly represented. The superstates are inferred by the hierarchy. As an example, considering the statechart of Figure 1, any configuration would have the general form $(H, G, C, B, (E, [(F12, F11), (F22, F21)]))$. A configuration with the states F11 and F21 active is noted by $(0, 0, 0, 0, (0, [(0, 1), (0, 1)]))$ or by $(0, 0, 0, 0, (0, [(0, F11), (0, F21)]))$.

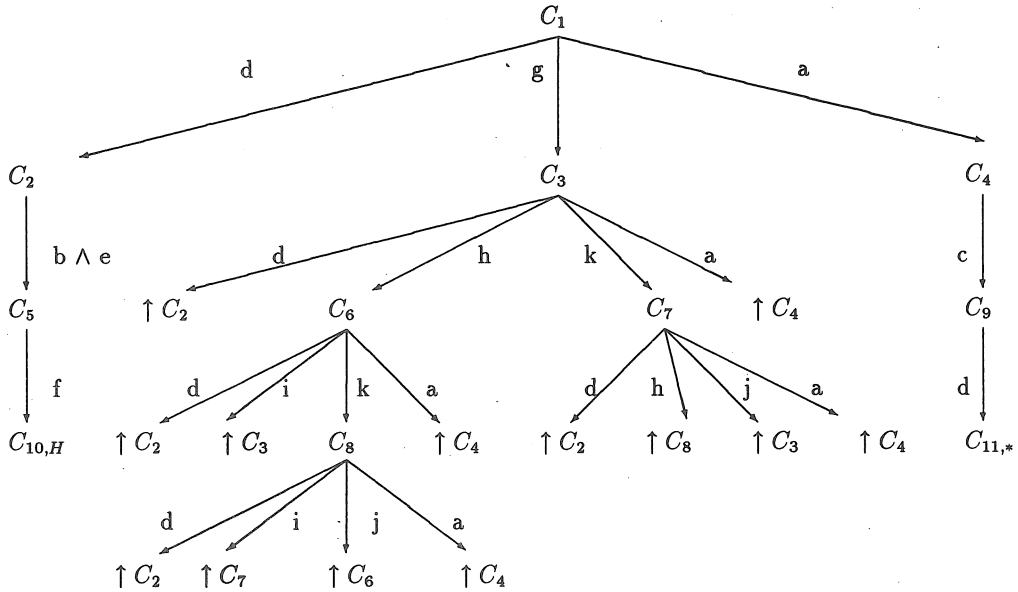
The reachability tree will be constructed considering every possible computation sequence. In each step we suppose that the event expression evaluates to true leading thus to a new configuration. Conditions are included in the event expressions and are also considered when firing a transition. The semantic rules for computing each step are the ones defined in [5]. The computation evolves in a sequence of time steps, which can be represented by the set $\{(\mathrm{SC}_i, \Upsilon_i, \Pi_i^*),\ i \geq 0\ \}$, where

1. $SC_0 = (X_0, \Pi_0, \Theta_0, \xi_0)$, where X_0 is the initial configuration and the other symbols represent the external estimuli occurring in the first time interval I_0. Π_0 is the external set of events generated by the environment, Θ_0 is the set of primitive conditions whose value is *true*, and ξ_0 is a function that for every variable x in the environment computes the value of x.

2. Υ_i is a step taken in SC_i, at the time step σ_{i+1}, $i \geq 0$.

3. Π_i^* is the set of events generated by Υ_i, $i \geq 0$.

We have then a sequence of state configurations, where SC_{i+1} is the system



Where the configurations are:

C =$(H, G, C, B, (E, [(F12, F11), (F22, F21)]))$

C1 =$(0,0,0,0,(1,[(0,0),(0,0)]))$
C2 =$(0,0,0,1,(0,[(0,0),(0,0)]))$
C3 =$(0,0,0,0,(0,[(0,1),(0,1)]))$
C4 =$(0,1,0,0,(0,[(0,0),(0,0)]))$
C5 =$(0,0,1,0,(0,[(0,0),(0,0)]))$
C6 =$(0,0,0,0,(0,[(1,0),(0,1)]))$
C7 =$(0,0,0,0,(0,[(0,1),(1,0)]))$
C8 =$(0,0,0,0,(0,[(1,0),(1,0)]))$
C9 =$(1,0,0,0,(0,[(0,0),(0,0)]))$
C10=$(0,0,0,0,(H,[(H,H),(H,H)]))$
C11=$(0,0,0,0,(*,[(*,*),(*,*)]))$

Figure 2: Reachability Tree for the Statechart of Figure 1

Valid Sequence of Events

A sequence of event expression is valid if each event may cause a transition to fire yielding a configuration change. Let $s_1 s_2 ... s_{n-1}$ be a sequence of event expression. The sequence will be valid if we can find a sequence of state configurations $C_0 \xrightarrow{s_1} C_1 \xrightarrow{s_2} C_2 \ ... \ \xrightarrow{s_{n-1}} C_n$ where s_1 can fire from C_1, s_2 can fire from C_2, and so forth. Usually, C_0 is the initial configuration. The algorithm will stop if for any $C_i, 0 \leq i < n$, there is not an arc labeled s_i with source in C_i. The algorithm will succeed in validating the sequence of events when the whole sequence $C_0 C_1 C_2 ... C_n$ is found. Let s_2= g, h, d, b ∧ e, f, i. It is a valid sequence of events and the corresponding state configurations are: $C_1 \xrightarrow{g} C_3 \xrightarrow{h} C_6 \xrightarrow{d} \uparrow C_2 \xrightarrow{b \wedge e} C_5 \xrightarrow{f} C_{10,H} \rightsquigarrow C_6 \xrightarrow{i} \uparrow C_3$.

The example above illustrates two points. First, it is shown that when we reach node ↑ C_2 we know it is not a terminal node and that it is linked to the first occurrence of the same configuration in the tree. We then proceed from it. For example, all configurations ↑ C_2 are linked to the configuration C_2 which is the leftmost son of C_1.

C_6 as indicated by the history configuration.

Reachability

We want to know if a certain configuration can be reached from another one. A reachable initial state configuration must be a reachability tree's node. Therefore, we search the configuration in the reachability tree and, if it is found, the sequences starting from the initial configuration are shown – they are the arc's labels in the path from the initial configuration to the target configuration. There may be many paths leading to the input configuration. If we use a breadth-first search strategy the shortest path will be found first. Others will be found, if there are any, continuing the search after the first occurrence is found. Let us consider the configuration with active state B. The configuration labeled (0,0,0,1,(0,[(0,0),(0,0)]) will be found in node C2. The path from the initial configuration to it is given by the event sequence <d>. Other paths can be found for this particular configuration, e.g., <g,d> and <g,k,d>.

Reachability from any configuration C_1 to another configuration C_2 is easily obtained if we: a) make sure both C_1 and C_2 are valid configurations; and b) build the reachability tree considering C_1 as the initial configuration. Then we have to find C_2 in the tree, as before. When there is history involved we will find the path leading from C_1 to C_2 always considering the first (default) visit to C_2.

Deadlock

A system modelled by a Statechart may reach a deadlock situation if its corresponding reachability tree has a terminal node. The converse is not necessarily true because a system may reach intentionally an end state as part of its specification. The algorithm looking for deadlocks has then to search the entire reachability tree looking for terminal nodes. They are output as possible deadlock configurations as well as the path from the initial configuration to the terminal node. History nodes are also leaves but as we always know how to proceed from them they are not considered terminal nodes. The example shown in this paper does not reach a deadlock configuration; examples can be found in [1].

Usage of Transitions

This property is verifyed looking for transitions which will never fire or maybe transitions left out of the specification (completeness). If this is the case, an arc labeled with the transition's label we are looking for will not appear in the reachability tree. Hence, the algorithm accepts as input an event expression supposed to be the label of a transition and searches for an arc with that label. If found, its source node (configuration) is output; otherwise, the transition is not used. As an example let us consider the transition labeled c. It is used and may fire from node C_4, whose configuration is (0,1,0,0,(0,[(0,0),(0,0)]).

4 Concluding Remarks

The algorithm for building the reachability tree as well as the ones for verifying properties were implemented and some testing conducted within an environment where there was already available some tools to support statechart specification and simulation: a graphical statechart editor, an analyzer for textual input, and a simulator [2]. The existence

```
01 BEGIN BuildSTR
02   Create an empty queue CQ
03   Make SC0 the root of the reachability tree STR
04   Insert SC0 in CQ
05   WHILE CQ is not empty
06     Remove SC from CQ
07     Let SC be the statechart configuration
08     Obtain SR, the set of all possible transitions for SC
09     IF SR is not empty THEN
10       FOREACH transition in SR
11         Let T be a transition in SR
12         Compute the step for T obtaining SC'
13         IF any destination state DS^k has a history symbol
                or its default state has a history symbol THEN
14           IF DS^k has not been active in this path THEN
15             Get DS_0^k, the default configurations for DS^k
16             Create a node with SC' replacing the subconfiguration of DS^k by DS_0^k
17             IF SC' exists in SRT THEN
18               Make node SC' an old configuration
19             ELSE
20               Make node SC' a new configuration
21               Insert SC' in CQ
22             ENDIF
23             Insert node SC' in SRT as a son of SC
24             Label the arc from SC to SC' as "<event-exp> /def"
25           ENDIF
26           Create a node SC'' from SC' replacing the DS^k subconfiguration by (H/*)
27           Make node SC'' a history configuration
28           Insert node SC'' in SRT as a son of SC
29           Label the arc from SC to SC'' as "<event-exp>"
30         ELSE
31           Create a node with SC'
32           IF SC' exists in SRT THEN
33             Make node SC' an old configuration
34           ELSE
35             Make node SC' a new configuration
36             Insert SC' in CQ
37           ENDIF
38           Insert node SC' in SRT as a son of SC
39           Label the arc from SC to SC' as "<event-exp>"
40         ENDIF
41         Reinstate SC as the statechart configuration
42       ENDFOREACH
43     ELSE
44       Make node SC a terminal configuration
45     ENDIF
46   ENDWHILE
47 END BuildSRT
```

Figure 3: Algorithm for Building a Reachability Tree

of these tools eased enormously the implementation effort and preliminary validation of these algorithms.

The many existing features in the statecharts forced us to define extensions to the conventional reachability trees, the major one being the history mechanism. Also, the mechanism for restoring a history configuration from the list of states in the search path is essential to representing history in the reachability tree. A trade-off decision had to be made as to represent explicitly or not the transitions leaving a superstate. We made the decision of representing it in each subconfiguration, trading the tree's size by having simpler algoritms for searching the tree. The algorithm for building the reachability tree would be much simpler if we had dropped history, as was done, for example, in [8], yet we believe the solution presented in this paper is simple and elegant. Without history we would not have available a powerful mechanism to represent concisely many recurrent system requirements.

References

[1] I. G. Boaventura. *Dynamical Properties of Statecharts* (in Portuguese). M.Sc. Dissertation, ICMSC-USP, São Carlos, 1992.

[2] R. P. de M. Fortes. *A Tool for Statechart Simulation* (in Portuguese). M.Sc. Dissertation, ICMSC-USP, São Carlos, 1991.

[3] i-LOGIX. The STATEMATE Approach to Complex Systems. 1989.

[4] D. Harel. STATEMATE: a Visual Formalism for Complex Systems. *Science of Comp. Programming*, 8 (3), 231-274, 1987.

[5] D. Harel. STATECHARTS: On the Formal Semantics of Statecharts.*Proceedings of the 2nd IEEE Symposium on Logic in Computer Science.* Ithaca, New York, 1987.

[6] D. Harel et alli. STATECHARTS: A Working Environment for the Development od Complex Reactive Systems. *Proceedings of the Tenth International Conference on Software Engineering.* (Singapure, April), Washington D.C.,IEEE, 1988.

[7] N. G. Leveson; J. L. Stolzy. Safety Analysis Using Petri Nets. *IEEE Trans. on Software Engineering*, SE-13 (3), 386-397, 1987.

[8] J. J. M. Hooman; S. Ramesh; W. P. de Roever. A compositional Axiomatization of Statecharts. To appear in *Theoretical Computer Science.*

[9] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Inc., Englewood Cliffs, N.J.,1981.

UN NUEVO METODO DE REPRESENTACION DE OBJETOS EN IMAGENES MULTINIVEL MEDIANTE UN GRAFO DIRIGIDO ETIQUETADO

Autores: R. Medina, A. Calvo, J.A. Romero, N. Fernández

Unidad Docente de Informática. Dpto. de Matemáticas Aplicadas.

Escuela Universitaria Politécnica. Universidad de Córdoba.

ABSTRACT.-

En el presente trabajo se desarrolla un nuevo método para representar objetos de una escena multinivel, obtenida tras un proceso previo de segmentación a bajo nivel. Las principales ventajas del mismo consisten, por un lado, en un bajo tiempo de cómputo derivado de la propia filosofía de su construcción, y por otro, en la enorme flexibilidad a la hora de obtener características de los objetos presentes en él. Dicho método ha sido comparado con un criterio clásico obteniéndose una mejora bastante sustancial y ha sido utilizado junto a un método nuevo para segmentar a bajo nivel escenas en color real [1], consiguiendo que la localización de especímenes dentro de preparaciones microscópicas sea posible en un tiempo reducido. [3]

KEYWORDS.- Estructuras de datos. Análisis de imagen. Segmentación a alto nivel.

INTRODUCCION.-

En el análisis de imagen, constituye una parte importante el proceso de segmentación. Dentro del mismo, pueden considerarse dos fases bien diferenciadas: clasificación de pixels en la imagen (dependiendo de criterios obtenidos por métodos supervisados o no) y agrupamiento de los mismos en regiones "conectadas". Los criterios utilizados a la hora de determinar las regiones presentes en una escena constituye lo que de una forma clásica se ha venido a denominar análisis de conectividad.

El propósito general de un análisis de conectividad, es conseguir aislar e identificar múltiples objetos (regiones con una determinada propiedad) en una imagen dada. La idea básica, subyacente en la descripción de un método de conectividad, consiste en asignar un identificador a cada objeto, de forma que cada pixel en la imagen pueda ser etiquetado con el identificador correspondiente del objeto al que pertenece. Los primeros trabajos sobre conectividad [5], basaron su estudio sobre escenas binarias, estableciendo como hipótesis de partida, el que cada objeto en la imagen está totalmente rodeado por otro con nivel de gris diferente.

Actualmente, existe una amplia bibliografía referente a métodos que hacen uso de técnicas de estructuras de datos para resolver problemas de conectividad [2][4][6][7][8]. Una de las técnicas clásicas consiste en detectar aquellos segmentos (el principio y fin), cuyo nivel de gris es "1" en la imagen binaria, obviando el resto. Partiendo de la hipótesis de que el fondo de la escena se encuentra a nivel cero, es evidente que de esta forma se pueden detectar en la escena diferentes objetos a través de sus segmentos.

Mediante un proceso basado en la comparación entre los extremos de diferentes segmentos (criterio de conectividad) , se pueden reconocer aquellos que pertenecen a un mismo objeto y en consecuencia se puede disponer de la información relativa a los objetos en forma de principio y fin de cada uno de los segmentos que pertenecen al mismo. En realidad, la información que está disponible

mediante este procedimiento, está constituida por "casi" todos los puntos del contorno del objeto, puesto que sólo aquellos puntos del contorno de un objeto que pertenezcan a un segmento de pendiente 0, estarán ausentes.(Fig. 1)

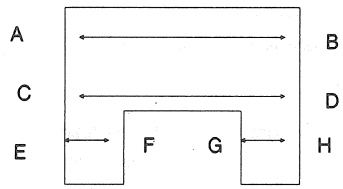


Fig. 1: Detección de segmentos pertenecientes a un objeto

La ventaja fundamental de este procedimiento, estriba en que simultáneamente al proceso de detección de segmentos en la escena, puede obtenerse información referente al tamaño, forma y posición de cada objeto. El principal incoveniente, consiste en que para la obtención del código de cadena del objeto, se requiere que durante la detección de los segmentos, se efectúe una comparación añadida, para unir aquellos extremos que deban enlazarse "ordenadamente" (Fig. 2). Este hecho, provoca que el método propuesto necesite un tiempo excesivo de cómputo, cuando en la escena existan varios objetos y se desea obtener el código de cadena de todos ellos, razón por la cual, tradicionalmente se ha preferido, para la obtención del código de cadena, el clásico procedimiento basado en la localización de un punto del borde, y a partir de él, encontrar los sucesivos puntos utilizando criterios de vecindad.

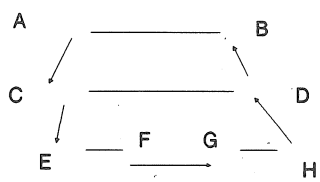


Fig. 2: Obtención del código de cadena para la fig.1

El propósito de nuestro trabajo ha ido encaminado a la obtención de un nuevo método, mediante el que se pueda construir, utilizando técnicas de estructuras dinámicas de datos, un grafo representativo de cualquier escena multinivel en el que no se necesite la obtención de ninguna característica durante el proceso de creación. Tal método presenta como ventaja fundamental el requerir un bajo tiempo de computación, puesto que cualquier característica puede ser obtenida posteriormente mediante un proceso de lectura eficiente, y permite que el mismo pueda integrarse dentro de procesos de control, que utilicen la visión como fuente de realimentación, en los que hay que tomar decisiones en tiempos reducidos .

METODO.-

El método que se propone consiste en la representación de los diferentes objetos en una escena multinivel, mediante un grafo dirigido etiquetado cuya construcción se basa en los conceptos de extremidades y relaciones padre-hijo. Dicho grafo, puede ser acíclico o poseer ciclos, dependiendo de los objetos encontrados en la escena. La mayor parte de las veces, los arcos entre los distintos nodos serán simples, pero puede ocurrir eventualmente que entre un par de nodos exista un arco doble.

En la fig.3 se muestra una representación gráfica de la forma típica que adopta el grafo que se propone en una escena multinivel. Como se observa existen 4 tipos de nodos diferentes etiquetados con L, S, I, y P. Los nodos de tipo L mantienen información general de los objetos integrados: identificador, nivel de grís etc....Los de tipo S contienen sólo el número de fila en el que el objeto posee segmentos, encontrándose en los de tipo I los extremos de esos segmentos. Por último, los nodos de tipo P contienen información relativa a la orientación de las relaciones padre-hijo detectadas y son útiles sólo para poder efectuar una forma de recorrido específica que proporcione el código de cadena.

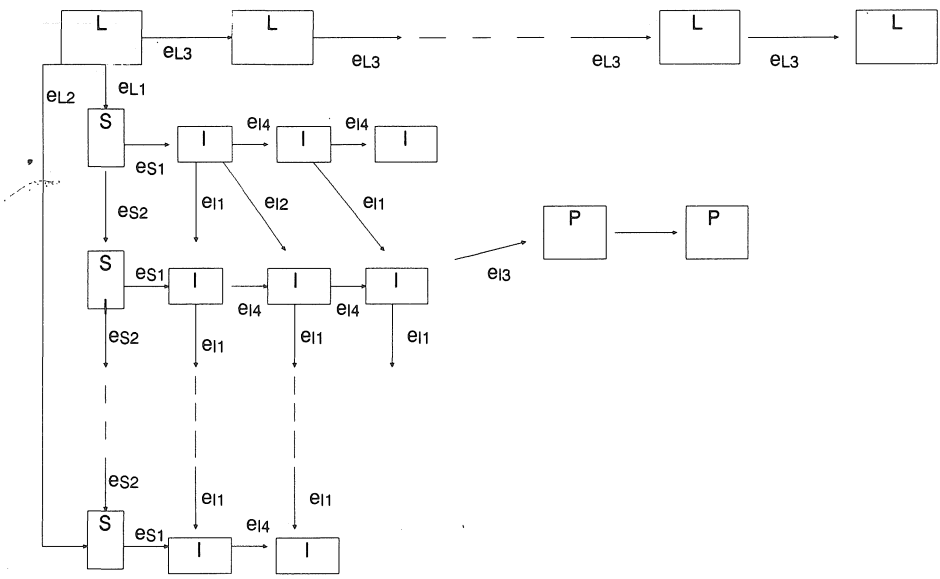


Figura 3:Representación gráfica del grafo propuesto.

Los enlaces presentes, son los necesarios para las 4 listas simplemente encadenadas en las que puede ser descompuesta la estructura general (cada una con nodos de un mismo tipo), y los que comunican los diferentes tipos de nodos.

Denominamos *Objeto* en una imagen multinivel a aquel conjunto formado por segmentos, pixels que estan a un mismo nivel de gris, con conectividad. El criterio de conectividad se basa en el solapamiento entre dos segmentos de filas consecutivas. Un objeto se dice que está activo mientras exista la posibilidad de que el mismo posea segmentos en filas posteriores. Este concepto es especialmente importante en relación a la velocidad de cómputo en la fase de creación, puesto que de no preveerse la desactivación, el número necesario de comparaciones crece de forma exponencial.

Diremos que un objeto posee *Extremidades*, cuando en su examen, algún segmento perteneciente a él, se subdivida en las filas posteriores en al menos dos segmentos, que también pertenezcan al mismo (Fig. 4). En un examen secuencial de arriba hacia abajo, un objeto será denominado *Hijo* de un *Padre*, cuando su inicio se encuentra con posterioridad al padre y se una a éste (Fig. 5).

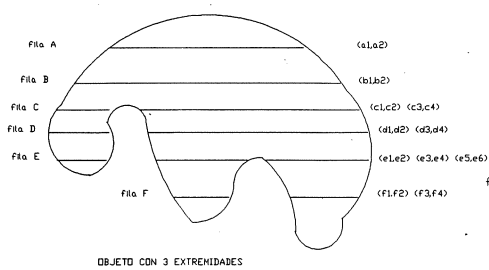


Figura 4: Objeto con tres extremidades.

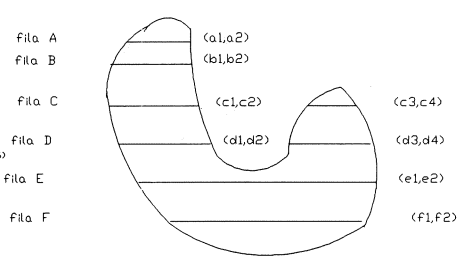


Figura 5: Objeto con un padre y un hijo a la derecha.

Supongamos un objeto que no posea extremidades ni relaciones padre-hijo (Fig 6). El grafo correspondiente al mismo tendría como representación gráfica la mostrada en la figura 7. En la misma puede observarse la existencia de un único nodo de tipo L y los enlaces entre los nodos de tipo L, S e I.

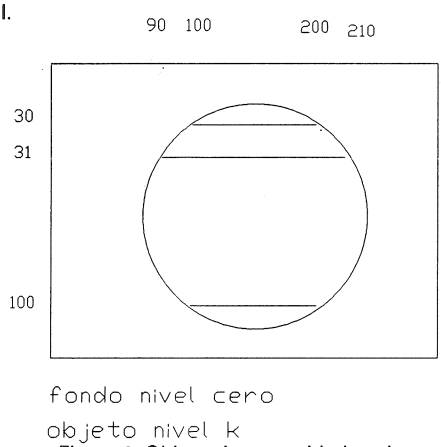


Figura 6: Objeto sin extremidades ni relaciones padre-hijo.

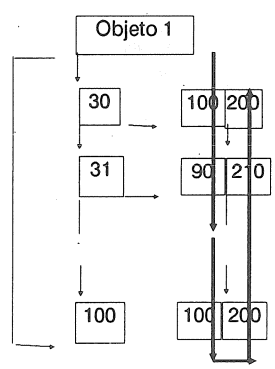


Figura 7: Grafo correspondiente a la figura 6

En el caso de que el objeto tenga *extremidades*, la diferencia se produce en los enlaces en diagonal que son los que se corresponden con su "aparición"(Fig.8) , mientras que las relaciones padre- hijo vienen dadas por los enlaces entre nodos de tipo I con otros de tipo L, además de la existencia de más de un nodo de tipo L para un mismo objeto(Fig.9).

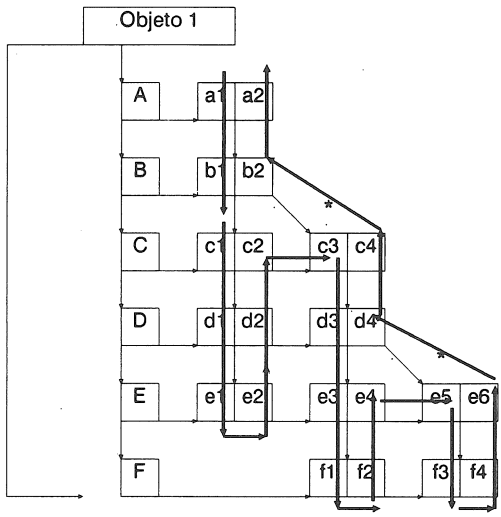


Figura 8: Grafo correspondiente a la figura 4.

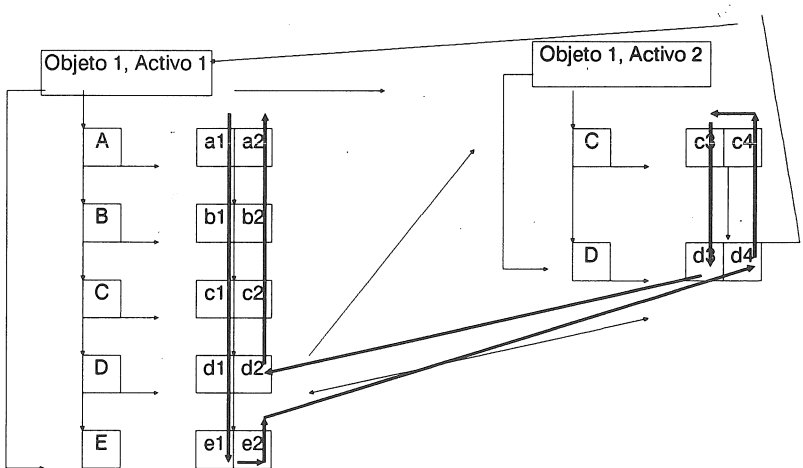


Figura 9:Grafo correspondiente a la figura 5.

Todo objeto en la escena queda caracterizado en el grafo mediante sus extremidades y relaciones padre-hijo, y cualquier característica deseada puede obtenerse con posterioridad, utlizando "formas" de recorrido en el interior del mismo. Para ello, si la característica deseada no requiere la obtención del código de cadena, puede realizarse un recorrido "secuencial" basado en la localización de un nodo de tipo L, en el que figura el identificador del objeto requerido, y a partir de él acceder a los nodos de tipo S, en los que se encuentran las filas en las que el objeto posee segmentos. Los extremos de estos se encuentran en los nodos de tipo I, donde es posible obtener cada pixel individual del objeto.

Obviamente esta forma de recorrido no es "ordenada", por lo que si se desea obtener el código de cadena, se requiere emplear un procedimiento diferente. Los conceptos de extremidades y relaciones padre-hijo constituyen información suficiente para tal propósito. Si se observa la figura 7, en la que el trazo grueso indica el recorrido necesario, solo es necesario implementar un proceso descendente simple y otro ascendente que debe ser recursivo en virtud de la orientación de los enlaces. En la figura 8, el recorrido ascendente debe ser interrumpido cuando se detecte la presencia de un enlace diagonal (*) para efectuar un proceso descendente, lo que obliga a efectuar un proceso recursivo múltiple. Una condición similar se requiere para el caso de relaciones padre-hijo, en el que deben ser detectados los enlaces entre nodos de tipo I y L(*) para interrumpir el recorrido del padre y proceder al del hijo en orden invertido. En este último caso, la orientación del hijo es fundamental puesto que los hijos a la izquierda deben ser recorridos en el proceso descendente, mientras que los hijos a la derecha deben serlo en el proceso ascendente.(Fig.9)

DISCUSION Y RESULTADOS.-

El método expuesto ha sido evaluado sobre un grupo de escenas en diferentes supuestos. En la figura 10 aparece en el eje de abscisas el número de objetos en cada escena y en el eje de ordenadas los tiempos de cómputo. Las lineas paralelas al eje de abscisas representan el tiempo que necesita el hardware de imagen utilizado (T) y la suma del mismo con la necesaria condición para detectar los posibles segmentos (T + búsqueda) en la lectura de una escena vacía. Su incorporación en la gráfica permite resaltar el tiempo que se atribuye en concreto al método implementado. Las restantes indican los resultados bajo las siguientes hipótesis: a) el fondo se asume a un nivel de grís conocido y no se integra en el grafo b) tiempo anterior más el necesario para obtener el código de cadena de todos los objetos c) no se conoce el nivel de grís del fondo y por tanto se recoge en el grafo como un objeto más y d) similar al caso b) pero integrando el fondo en el grafo.

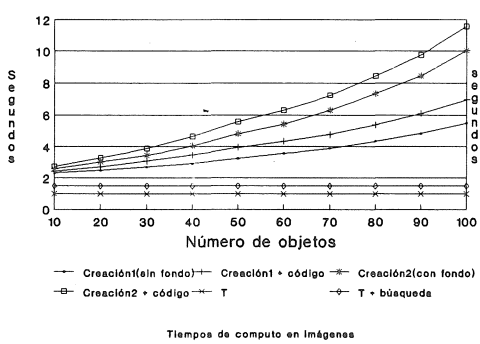


Figura 10: Tiempo de cómputo en diferentes supuestos

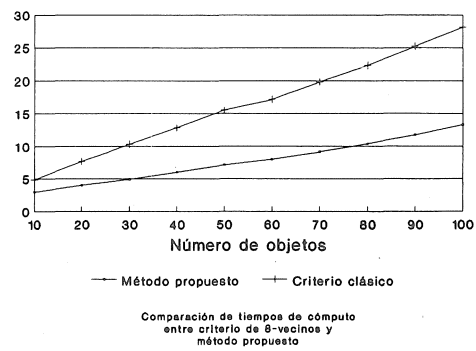


Figura 11: Método que se propone frente a criterio clásico

En la figura 11 aparece la comparación efectuada entre el método que se propone con el criterio clásico de vecindad, empleado usualmente para obtener el código de cadena. El mismo, tiene la limitación, de tener que modificar el nivel de gris de cada pixel del borde encontrado, a un nivel diferente al del objeto y el fondo, antes de encontrar el siguiente. Por el contrario, con el método que proponemos, tal limitación no existe, pero a efectos de que la comparación pueda ser efectiva, hemos forzado a que nuestro método "dibuje" el código de cadena de cada objeto. Ambos métodos han sido comparados sobre un grupo de escenas con diferente número de objetos. Como puede apreciarse en la representación, existe una relación 1 a 2 favorable al método diseñado.

Las principales ventajas del método propuesto pueden resumirse en los siguientes puntos:

1.- Representar objetos de una imagen, previamente segmentada, que no sea estrictamente binaria (imagen multinivel), lo que quiere decir que un objeto no tiene por qué estar totalmente rodeado por otro de nivel de grís diferente.

3.- La creación del grafo es razonablemente rápida, puesto que durante la misma no se calcula ninguna característica. En otros procedimientos se requieren comparaciones adicionales cuando se desea obtener el código de cadena de los objetos.

4.-La extracción de características clásicas (área, código de cadena, perímetro etc ...), se realiza mediante el uso de algoritmos específicos de lectura en el grafo, lo que implica una menor dependencia del hardware que se maneje.

5.- Los conceptos de extremidades y relaciones padre-hijo, bajo cuya filosofía se genera el grafo, constituyen en sí mismas nuevas características, invariantes a traslaciones y homotecias, útiles para determinados propósitos (por ejemplo reconocimiento de señales de navegación en robots móviles).

6.- Es posible la eliminación de objetos, basada en cualquier característica que se desee, durante la creación del grafo, puesto que siempre se conoce la terminación en el instante en que se produce. Este hecho es útil por ejemplo en la eliminación del ruido inicialmente existente en la imagen y que usualmente se traslada en el proceso de segmentación a bajo nivel. El procesamiento de los huecos de los objetos es extremadamente sencillo, puesto que forman parte del grafo construido y su detección es rápida mediante una lectura secuencial. Esto permite la obtención de parámetros topológicos de forma simultánea. Los hechos citados hacen que pueda ser un método alternativo a las operaciones de erosión-dilatación, muy lentas si no se dispone de un hardware específico.

La "robustez" del método propuesto ha sido comprobada en un proyecto destinado a hacer posible la total automatización en un proceso de reconocimiento de fases del ciclo celular. Para ello, se han combinado técnicas de control automático sobre instrumentación, con procedimientos de análisis de imagen en color real [1] permitiendo diseñar un sistema totalmente autónomo, que únicamente requiere la presencia del investigador para la colocación de la preparación sobre la plataforma móvil del instrumento utilizado [3].

AGRADECIMIENTOS.-

Este trabajo está financiado en parte por el proyecto de la C.I.C.Y.T. ROB90-0992-CO3-02.

BIBLIOGRAFIA.-

[1] A. CALVO, y R. MEDINA " A new method of segmenting real color scenes applicable to automatic control processes". 23 rd International Symposium on Industrial Robots. Barcelona October 1992.

[2] T.S. HUANG, "Image sequence processing and dynamic scene analysis". Springer-Verlag. 1983.

[3] R. MEDINA y A. CALVO "Automatic control of a citophotometer trough visual feedback with biomedical applications". IFAC Symposium on Intelligent Components and Instruments for Control Applications. págs 505- 510 Málaga. 1992.

[4] Y. OHSAWA y T. SAUCHI. "The BD- tree. A new N-dimensional data structure with highly efficient dynamic characteristics". Information Processing. R.E.A. pp. 539-545. 1983.

[5] A. ROSENFELD, "Secuential operations in Digital Picture Processing". Journal of the ACM. vol 13 pp 471-494.

[6] A. ROSENFELD y R. HUMMEL, "Scene labeling by relaxation operations". IEEE Ttransactions on Systems, Man and Cybernetics. vol 6 pp. 420-423. 1984.

[7] SHI-KUO-CHANG "Principles of pictorial information system design". Prentice-Hall 1989.

[8] H. WECHSLER, "Computational vision". Academic Press. 1990.

El SETDC: un sistema basado en el modelo TDC.

Lic. José E. Medina Pagola
Profesor Auxiliar

Ing. Giselle Castro Dorticós
Profesor Instructor

Ing. Lisset Alfaro Remesal
Profesor Instructor

Departamento de Informática
Facultad de Ingeniería Industrial
Instituto Superior Politécnico José Antonio Echevarría
Carretera Central Manuel Martinez Prieto
Marianao. Ciudad de la Habana, Cuba.

El SETDC: un sistema basado en el modelo TDC.

Los elementos esenciales de los sistemas expertos (SE) son la base de conocimientos y la máquina de inferencias. A través de ellos los ingenieros en sistemas pueden desarrollar aplicaciones en las que las informaciones se representen de una forma más declarativa y en las que sean posibles inferir nuevas informaciones a partir del conocimiento representado.

Los SE están compuestos no solo de los elementos anteriores, sino que también pueden incluir, a partir de las secuencias deductivas desarrolladas, la habilidad de explicar el razonamiento seguido que justifica las soluciones tenidas, de una forma cercana al razonamiento que seguirían los expertos de esa actividad.

Las características antes mencionadas, unido a las posibilidades de representar el conocimiento mediante una lógica polivalente, hacen posible el desarrollo de aplicaciones en variados campos, tales como:

- el diagnóstico de fallas
- el pronóstico de diversos procesos
- la organización y planificación inteligentes de múltiples actividades
- la enseñanza o tutoría inteligente
- etc

El SETDC es un sistema vacío que permite desarrollar diversas aplicaciones afines con los SE utilizando como esquema de diseño al modelo TDC.

En este trabajo se exponen las características fundamentales del modelo TDC y su concresión en este sistema, así como algunas consideraciones en su aplicación y su desarrollo ulterior.

El modelo TDC

El modelo TDC (de Tablas de Decisión Categorial) es un esquema conceptual para la representación y manipulación del conocimiento. (1 y 2)

En su concepción este modelo incluye elementos de otros, tales como el cálculo de predicados, los basados en reglas de producción y las redes semánticas. Sin embargo, el modelo TDC no se reduce a ellos ya que este integra a aquellos de una forma propia, enriqueciéndolos con nuevas estructuras y mecanismos.

La base de conocimientos en este modelo es representada mediante tres elementos básicos:

- la red taxonómica o jerárquica de categorias
- el conjunto de hechos
- las tablas de decisión

Todos estos elementos se integran en un mecanismo inferencial de backward chainning perfeccionado con procesos específicos de case y procesamiento conjunto de antecedentes de tablas de decisión, denominado también como proceso alfa.

Estos elementos y otros específicos serán detallados al describir el sistema SETDC.

El SETDC 1.0

El SETDC 1.0 es la primera versión realizada sobre el modelo TDC (3). Este sistema está conformado por los siguientes subsistemas:

- el generador de la base
- el monitor de la base
- los utilitarios del sistema

El generador (GENTDC) es un programa capaz de procesar la descripción de una base de conocimientos editada como un fichero ASCII, analizar sintácticamente dicha descripción y generar la base de conocimientos que será efectivamente procesada por la aplicación.

El monitor de la base (MONTDC) permite verificar el correcto procesamiento de la base generada por GENTDC. Mediante este programa no solo se pueden analizar las inferencias a obtener y el funcionamiento de cada utilitario del sistema, sino que también puede comprobarse paso por paso cómo se efectúa el proceso inferencial con la base dada.

Los utilitarios del SETDC son un conjunto de procedimientos que desarrollan todas las facilidades del sistema. Estos están organizados en units del Turbo Pascal 5.5.

De lo anterior se infiere que las aplicaciones finales pueden programarse en dicho lenguaje, asumiendo las interfaces deseadas y garantizando diseños con toda la profesionalidad posible.

Para el empleo de estos subsistemas debe seguirse los pasos que se indican en el siguiente esquema:

En este esquema no solo se indica la interrelación entre estos tres subsistemas, sino que además se expresa el carácter de prototipo y de perfeccionamientos sucesivos (a) que está presente en la creación de todo sistema informativo (4) y de todo SE (5).

La descripción de la base de conocimientos

Como fue mencionado anteriormente, la base de conocimientos es descrita mediante cualquier editor de texto utilizando un lenguaje específico.

En dicho lenguaje se reconocen 4 partes denominadas también como bases, las cuales son:

- la base de átomos.
- la base de predicados.
- la base de hechos.
- la base de reglas.

En la base de átomos se definen todas las categorías atómicas a ser utilizadas en la descripción del conocimiento, así como las relaciones que se establecen entre ellas.

Un átomo es una categoria simbólica que denota algún concepto y que puede relacionarse con otras categorias mediante relaciones de ordenamiento, denominada en el modelo TDC como relaciones de afinidad. Por ejemplo, si se definiera la siguiente base de átomos:

```
ATOMOS
raúl:talentoso;
raúl:alumno_instructor;
talentoso, alumno_instructor:alumno;
centro_de_investigación:centro;
```

se estarían definiendo seis categorías atómicas con cinco relaciones de afinidad explícitas entre ellas.

Tales relaciones de afinidad pueden definirse no solo en forma arbórea, como puede apreciarse en el ejemplo, sino también en forma de grafo, estableciéndose una red taxonómica o jerárquica entre las categorías atómicas.

Esta característica posibilita su manipulación a través de un proceso de herencia múltiple, permitiendo definir intersecciones o conjunciones semánticas de los conocimientos heredados.

En el modelo TDC, además de las categorías atómicas, se tienen las categorías de objetales, aunque en esta primera versión estas últimas no se han incluído.

En la base de predicados se definen todos los predicados que utiliza la base de conocimientos. En la definición de cada predicado se indica no solo los argumentos que estos permiten, sino también las categorías atómicas superiores en ordenamiento admitidas por esos argumentos.

A modo de ejemplo supóngase la siguiente base de predicados:

```
PREDICADOS
responsable(alumno);
puede_trabajar(alumno,centro);
  .
  .
```

En esta base se definen dos predicados, cada uno con uno y dos argumentos y tales átomos como categorias superiores permitidas para esos argumentos.
Además de las informaciones antes descritas, cada predicado admite la definición de diversos tipos, los cuales serán analizados junto a los procesos inferenciales incluidos en el sistema.

La base de hechos define todas las informaciones incondicionalmente ciertas o falsas que servirán como hechos de partida en el SE que se diseñe.

Los argumentos permitidos en los hechos pueden ser constantes atómicas o los tipos Pascal longint, real, boolean y string, garantizando con ello una mejor compatibilidad con los procesos a diseñar en Turbo Pascal.

Las constantes atómicas pueden cuantificarse universal (V) o existencialmente (E) de la siguiente forma:

p(a) se corresponde con: V a p(a)

p(a^) se corresponde con: E a p(a)

no p(a) se corresponde con: V a ~p(a)

no p(a^) se corresponde con: E a ~p(a)

siendo "a" un átomo cualquiera.

Nótese que la cuantificación de un término atómico se realiza en el propio argumento, lo cual, aunque simplifica su notación, dificulta la modificación del orden de cuantificación entre los términos de un mismo predicado. Esta limitación, además de ser soluble, no repercute significativamente en los SE que se realicen, pues es comprensible que el uso de la cuantificación existencial resulta poco frecuente.

Tanto la base de hechos como la de reglas permiten definir valores de certezas asociados con las informaciones contenidas en dichas bases.

Estos valores de certezas se toman en el intervalo [-1.0,1.0], considerando sus extremos lo absolutamente falso y cierto y su centro (0.0) lo absolutamente desconocido.

A modo de ejemplo supóngase la siguiente base de hechos:

```
HECHOS
responsable(alumno_instructor)/0.8;
```

En esta base se ha definido un único hecho, el cual expresa con una certeza de 0.8 que todos los alumnos instructores son alumnos responsables.

En la base de reglas se definen las reglas de producción agrupadas en tablas de decisión (TD).

Según el modelo TDC, una TD contiene los siguientes elementos:

	Reglas
Condiciones	Estados condicionales
	Certeza
Acciones	Estados de las acciones

Siendo las condiciones y acciones de la tabla los antecedentes y consecuentes de las reglas y los estados de cada una de ellas la forma en que se incorporan tales elementos para cada regla.

En el modelo TDC se permiten dos formatos de estados condicionales, correspondiéndose con las tablas de entradas limitadas y ampliadas. En esta versión del SETDC se permiten diseñar solamente TD de entrada limitada.

Una TD de entrada limitada es aquella que admite como estados condicionales tres categorias básicos: lo cierto (si), lo falso (no) y lo desconocido (*), además de permitir cualquier combinación de tales estados.

Las categorías anteriores establecen los valores de certezas aceptados por cada regla para las inferencias asociadas con cada antecedente. Para ello, tales valores de certezas son clasificados en tres subintervalos de la siguiente forma:

umbral < certeza <= 1.0 para lo cierto

-umbral <= certeza <= umbral para lo desconocido

-1.0 <= certeza <= -umbral para lo falso

En donde el valor de umbral es un valor modificable por el usuario y asumido de forma estandard como 0.2.

Al igual que los hechos, los antecedentes y consecuentes de tablas admiten cualquier constante de las asumidas en el sistema como válidas. Sin embargo, en los predicados definidos como antecedentes o

consecuentes también se permiten argumentos variables.

Un argumento variable sigue el formato:

a # N

donde:

a: es un átomo cualquiera que define la categoría superior admitida por ese argumento.

N: es un entero positivo o cero.

Por ejemplo, si se definiera la siguiente base de reglas:

```
REGLAS
TABLA ejemplo
responsable(talentoso#)                                    ¦si;
CERTEZA                                                    ¦0.9;
puede trabajar(talentoso#,centro_de_investigación)¦*;
```

entonces se habría definido una TD de una sola regla, la cual indica, con una certeza de 0.9, que aquel alumno talentoso que sea responsable puede trabajar en cualquier centro de investigación, por lo que se pudiera inferir, con la base de átomos y de hechos ejemplificada antes, que raúl puede trabajar en tales centros con cierta certeza.

El proceso inferencial

En el SETDC 1.0 los predicados a ser verificados inicialmente son los antecedentes de alguna tabla que se lanza como TD raíz del proceso. Las proposiciones inferidas a través de los consecuentes de la tabla raíz son almacenadas en bases de hechos definidas como de inferencias. Los antecedentes de la tabla raiz son verificados mediante una búsqueda en backward chainning.

O sea, todo antecedente trata de verificarse con alguna proposición de la base de hechos o la considerada como de inferencias, en dependencia del tipo de predicado asociado con el antecedente y definido en la

base de predicados. Si un antecedente no puede verificarse con un hecho, entonces trata de verificarse con algún consecuente de TD, la cual a su vez requiere verificar nuevos antecedentes, repitiéndose este proceso hasta la verificación total de los antecedentes. Si durante este proceso algún antecedente fallara, se analizarían los mismos mediante el proceso conocido como backtracking.

Además de estos mecanismos ampliamente conocidos, el modelo TDC incluye otros procesos específicos que complementan los anteriores: el case y el procesamiento conjunto de antecedentes o proceso alfa.

El case y el proceso alfa consideran las peculiaridades que del modelo TDC se han implementado, tales como:

- la existencia de una red jerárquica de categorías atómicas.
- la posibilidad de cuantificar universal o existencialmente dichas categorías.
- las inferencias lógicas y de herencia que se establecen por esos conceptos.
- el encapsulamiento y comunidad de antecedentes de las reglas contenidas en cada tabla.

En todo este proceso inferencial se implementarion funciones de propagación de los valores de certezas, las cuales se corresponden con el modelo teórico desarrollado por Hájek (6).

Además de los procesos inferenciales concebidas en el modelo TDC, este sistema permite implementar en Turbo Pascal procesos asociados con algunos predicados cuyos tipos fueron definidos como tal en la base de predicados. Estos procesos asociados son invocados al requerirse la verificación de un antecedente cuyo predicado sea de ese tipo.

De esta forma, el ingeniero de conocimientos pudiera diseñar bases que contemplen otros procesos específicos de su aplicación.

Consideraciones finales.

Como fue planteado al inicio, el SETDC 1.0 es una primera aproximación al modelo TDC, en la cual ha sido implementado algunas de las características esenciales de ese modelo.

A pesar de la corta existencia de este sistema, se ha estudiado en un plano teórico el diseño de tres SE diferentes para:

- complementar procesos de asignación de recursos en una empresa de transportación.
- diagnosticar las posibles causas ante fallas en equipos y componentes de aviación.
- realizar la selección de rodamientos.

Este último SE se encuentra en la actualidad en las últimas fases de su implementación.

Como resultado de estos trabajos se ha verificado no solo la viabilidad del modelo y el sistema, sino que además ha corroborado las bondades esperadas.

Entre ellas debe resaltarse las ventajas que ofrecen las TD, tanto en su ejecución como en su carácter de instrumento de diseño y comunicación con los expertos, coadyuvando al completamiento y perfeccionamiento de las reglas y del conocimiento en general.

Tales trabajos también han permitido verificar las posibilidades de las categorias atómicas y su red taxonómica, aportando nuevos elementos los cuales se están analizando de una forma integral en sus dimensiones teórico-metodológicas.

Sin embargo, no obstante las posibilidades qué ofrece esta primera versión, se ha comprobado la conveniencia de implementar nuevas

versiones del SETDC conteniendo otras características no contempladas del modelo TDC, tales como las categorias de objetales y las TD de entrada ampliada.

Los trabajos realizados y otros iniciados en la esfera del control de procesos tecnológicos han planteado nuevas exigencias al sistema como es la inclusión de procesos asociados no solo con antecedentes sino también con consecuentes, lo cual permitirá realizar SE con mayores posibilidades, requiriéndose quizás en un futuro próximo la normalización de tales procesos (7).

Muy asociado con lo anterior, se ha concebido la necesidad de desarrollar y perfeccionar, según las características del modelo, procesamientos forward-backward, lo cual potenciaría las posibilidades y tipos de aplicaciones a realizar en este sistema.

Todas estas características y otras se encuentran en un proceso de estudio para ser incluídas en futuras versiones del SETDC.

Bibliografia.

(1) Medina P., José E. "Sistemas de Tablas de Decisión". Ponencia de Informática 88. La Habana, 1988.

(2) Medina P., José E. "El modelo de Tablas de Decisión Categorial: una representación del conocimiento en los sistemas expertos". Monografía. Rev. Ing. Industrial, Cuba (en proceso de edición).

(3) Medina P., José E.; Castro D., Giselle y Alfaro R., Lisset. "SETDC 1.0 Manual de Usuario", 1990.

(4) Marco, Tom de. "Structured Analysis and System Specification". Ed. Revolucionaria, La Habana, 1986.

(5) Frenzel Jr. Louis E. "Crash Course in Artificial Intelligence and Expert Systems" S/A.

(6) Hájek, P. "Combining functions for certainty degrees in consulting systems". Inter. Journal Man-Machine Studies. Academic Press Inc., London, 1985.

(7) Díaz, Oscar; Jaime, Arturo. "Un gestor extensible de procedimientos asociados en el contexto de los sistemas orientados a objetos" IBERAMIA 92 La Habana, Cuba, Ed. Limusa, México, pp 59-74, 1992.

SIRNEM: A Parallel Neural Network Simulation System

Wagner Meira Jr * Marcio L B Carvalho †

March 10, 1992

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Caixa Postal 702
30.161 - Belo Horizonte - Minas Gerais
Tel: +55-31-443-4088
Fax: +55-31-443-4352

Abstract

In this article, we present SIRNEM, a system to simulate the behavior of neural networks using a message-passing multicomputer. The problem of designing an efficient inter-neuron communication algorithm is addressed, this algorithm should also enable a reliable exchange of information, avoiding unpredictable situations as deadlock and starvation. A brief description of the main modules of SIRNEM are presented.

1 Introduction

In the context of computer science, the expression neural networks has been used to describe systems that have a great number of simple processors. These processors communicate among themselves by sending their activation values to every processor it is connected to; these values are also called the state of processor. A weight is associated to each connection, showing how significant the transmitted information is. The behavior of these systems are a function of the interaction between the processors through their connections. A motivation to investigate these systems comes from the analogy between them and the human brain.

Although there is available some theoretical development about neural networks, they are not sufficient to the completely describe the behavior of these systems, so the technique most frequently used to understand the behavior of neural networks is simulation. As in other computational models, it is important that researchers can implement and test their ideas. There is a great number of models of neural networks, each of them developed for different specific applications, when using neural networks to a new application it is necessary to simulate the models. There are not, until nowadays, general use neural computer so parallel simulation of neural networks is a "realistic" good and cheap way to study these systems.

An important feature of the neural networks is that they perform a great number of parallel computations [FH83], memory is distributed, processing is asynchronous and systems have a natural fault tolerant capacity [HMT86]. Parallel neural network simulation is much faster and realistic than the sequential one. But a distributed simulation comes with problems inherent to network models particularities.

Writing simulation programs for these models is a hard and error-prone work mainly because of the difference between usual programming languages primitives and the basic operations of neural networks. These simulations have great computational cost and a way to minimize it is to explore the parallelism of the neural network systems in a parallel simulation.

*E-mail:MEIRA@DCC.UFMG.BR
†E-mail:MLBC@DCC.UFMG.BR

In this article, we present an environment for simulation of different models of neural networks in a multiprocessing architecture. In the next section, the question of the inter-neuron communications reliability is studied. In the third and fourth sections we describe the simulation of neural networks; in the third we present how we can specify a neural network for simulation and in the fourth how we perform a simulation with SIRNEM.

2 Inter-neuron communications

Neural networks are discrete dynamical systems whose behavior is completely specified in terms of a local relation; time advances in discrete steps and the laws for these cells are expressed by a single recipe through which at each step each neuron computes its new state from that of its neighbors.

We identify an important property of neural network: The cells of the network work synchronously, at each step all the neurons must obtain the state of their neighbors to determine their next state. To correct simulate a neural network, the inter-neuron communications must be performed in such a way that guaranties the maintenance of the above mentioned property. We will now discuss some alternatives to solve this problem:

Establishing a synchronizing process may be the simplest solution to this problem; this process waits until all the neurons are ready and only then permits that they change their states [FFGL88]. A disadvantage of this policy is that it creates a bottleneck, limiting the parallelism of the system.

Another solution is to implement the synchronization in a distributed way [BL90], by the neurons; it is called implicit synchronism. This option is better than the previous because it allows more parallelism and neurons that do not have dependencies, may operate in parallel. A problem that arises is to determine what neurons may do so. To solve it we need to study the communications in a neural network and what conditions are necessary to satisfy the property described above.

We now present a solution that deals the synchronization of simulating asynchronous operation of neural networks, which can be found in the literature [GB81, CM84, BG89]. We will give an overview of this solution, but, before, we need to define some terminology:

The neural network may be viewed as a directed graph where the vertices are associated to the neurons and the edges to the connections between the neurons. The direction of each edge is the same as the information flow. So, there is an edge from vertex i to vertex j, if the state of neuron i influences neuron j.

It is possible that two neurons mutually influence each other. In this case there is an edge from i to j and another one in the reversed direction ($j - i$). An example of this case is the Hopfield's neural networks [HT86], where each neuron is connected with every other in the network. We observe that for each pair of connected neurons, only one of then may operate in the same step; if both were allowed to operate, there could be unpredictable situations as deadlock, starvation or an inconsistent network.

We can classify the communication between any two neurons (i and j) of the network in two types:

1. Bidirectional: there is an edge from i to j and another from j to i;
2. Unidirectional: there is an edge from i to j and there is not an edge from j to i;

The algorithm now being described makes the following assuption about the neural network to be simulated:

- The neural network to be simulated has only bidirectional connections.
- Each discrete step is composed by one or more substeps, and in each of them a group of independent neurons obtain activation values from neighbors. Because information flow through each connection only in one direction at a given substep, the direction of each

neuron to neuron communication in a substep k can be represented by a directed edge and the graph formed by these edges is the subgraph G_k, where there are only one edge between each connected neurons.

If G_k is acyclic, there can not be any unpredictable situations in the information exchanges of the network. It is shown in [GB81, CM84, BG89].

The main aspect of this solution is that it guaranties that each neuron communicate to its neighbors in a step. This can be achieved by maintaining G_k acyclic during the evolution of the network. If this condition is satisfied, there is at least a sink [1] node in G_k that has received all the information it needs, so it can calculate its next state.

After the sink has calculated its next activation value, this value can now be sent to its neighbors so they can calculate their new states too. As each sink computes its new state, it does not need any more information about the current step, so it reverses its edges and turns into a source.

At each substep, at least a neuron turns into a source and after all the neurons have turned into a source a step has been completed.

There are some neural networks with neurons that have bidirectional and unidirectional connections (we call them hybrid neurons) [BN89]. If this class of neural network uses the algorithm described above, it can cause unpredictable situations because the direction of some of its connections must be reversed and others can not to complete a step.

We propose an extension to the previously presented algorithm to simulate hybrid networks as well. In our solution, a neuron that have unidirectional connection will only change its state when it has turned into a sink and has received all the information from its unidirectional connections. Neurons with only bidirectional connections are treated in the same way as the original algorithm.

As we have some fixed direction edges, the algorithm is not able to guarantee that every instantaneous subgraph G_k is acyclic. So we must verify, from the definition of the network's topology, whether there is the possibility of cycles that include a fixed direction edge. A way to detect this kind of cycle in a graph G is verifying if the graph has a strongly connected component [2] which contains a fixed direction edge, which is the only situation that may have cycles.

3 Neural network simulation

We can identify two main activities in a neural network simulation: Initially we must define the model to be simulated and using it, we perform simulations, applying different inputs to the network and verifying its outputs.

When we are simulating these systems, a difficulty that arises is the great functional and topological differences among the models. It justifies the necessity of a flexible way to describe and simulate neural networks.

Another important feature to this kind of simulation is its homogeneity. Usually, the neurons of a network are functionally identical. We specify each neural network in three levels of hierarchy:

Neuron: a type of a neuron is based on a functional specification of a defined cell, its definition is completed with the exactly number of inputs, outputs and parameters.

Cell: in this level, we define how a cell of the network will work. This is done by some C code. Each cell must receive as parameters its number of inputs, outputs and some initial parameters.

Network: a neural network is composed by some interconnected neurons. In this level, must be defined the type of each neuron, its connections for input and output and any parameters it needs to work.

[1] **Sink:** a vertex were all edges are directed inward.

[2] **Components:** set of vertices of a graph with all the edges that connect any two vertices of the set. **Strongly connected components:** component where there are two paths between any two vertices. [Szw84]

From one neural network definition we may need to execute a large number of various simulations. Each of them may contain its own features, so it would be usefully a flexible way to specify the simulations is available and interact with the network to obtain data to perform the simulation.

We then establish that the simulator must be also defined in the network specification phase, with all of its inputs and outputs defined and connected. The dealing of information exchanged between the simulator and any neuron must be done in a separate way from the inter neuron. These connections must be dealt in a explicit way in the neuron definition and declared with the type of the neuron.

4 SIRNEM

SIRNEM is an environment for simulation and study of neural networks. The user must specify how his network works (as outlined above) to generate a simulation. The environment deals with aspects like synchronization, allocation of processes into processors and the application generation.

To simulate a network, the user perform the following steps:

1. Definitions of the cells;
2. Definitions of neurons;
3. Definition of network;
4. The C code for the application is generated assuming one process per neuron and one for simulator. All the inter-process connections are specified. Code generation and the following operation are performed simultaneously:
   - It is verified whether the network has potential cycles, as described in section 2.
   - The neuron processes are allocated to the available processors, to obtain an even load balance. As all processes have basically the same computational cost, this embedding problem can be modeled as a graph partition problem as presented in [LFT92]. In a system of k processors, G must be partitioned in k subgraphs with a minimum number of connections among subgraphs. This problem is NP-complete and our solution uses Hopfield's neural networks as described in [LFT92].
5. All the code generated is compiled and the resultant application is loaded and executed in the target machine.

SIRNEM is implemented using facilities provided by another environment [MJSV+92] that solves some communications restrictions and allows all processes to use I/O resources.

By the time we are concluding this article, SIRNEM is already implemented and is being tested. We will include simulation results in the full version of the papers.

5 Conclusion

It is presented an algorithm to avoid unpredictable situations like deadlocks and starvations in a parallel neural network simulation. This algorithm can easily be extended to simulate not only neural networks but also any kind of cellular automata.

A neural network simulation environment is also presented. To perform the simulations, it needs only the description information about the network to generate a complete application program. This application has included the described protocol and other features as a balanced allocation of processes on processors and tools to monitor all the work of the neural network.

References

[BG89] V.C. Barbosa and E.M. Gafni. Concurrency in heavily loaded neighborhood-constrained systems. *ACM Transactions Programming Languages and Systems*, (11):562–584, 1989.

[BL90] V.C. Barbosa and P.M.V. Lima. On the distributed parallel simulation of hopfield's neural networks. *Software Practice and Experience*, C-20(10):967–983, October 1990.

[BN89] R. Balaniuk and P.O. Navaux. Reconhecimento de padrões codificados em seqüências utilizando redes neurais. In *Anais do Sexto Simpósio Brasileiro de Inteligência Artificial*, pages 359–372, Novembro 1989.

[CM84] K.M. Chandy and J. Misra. The drinking philosofers problem. *ACM Transactions Programming Languages and Systems*, (6):632–646, 1984.

[FFGL88] J.A. Feldman, M.A. Fanty, N.H. Goddard, and K.J. Lynne. Computing with structured neural networks. *Communications of the ACM,*, C-31(2):170–187, February 1988.

[FH83] S.E. Fahlman and G.E. Hinton. Massively parallel architectures for ai: Netl, thistle and boltzman machines. In *Proc. National Conference on Artificial Inteligence*, pages 109–113, 1983.

[GB81] E.M. Gafni and D.P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, (29):11–18, 1981.

[HMT86] D. Hammestrom, D. Maier, and S. Thakkar. The cognitive architecture project. *Computer Architecture News*, (14):9–21, 1986.

[HT86] J.J. Hopfield and D.W. Tank. Computing with neural circuits: a model. *Science*, 233:625–633, 1986.

[LFT92] K.C. Lee, N. Funabiki, and Y. Takefuji. A parallel improvement algorithm for the bipartite subgraph problem. *IEEE Transactions on Neural Networks*, C-3(1):139–145, January 1992.

[MJSV+92] W. Meira Jr, J.P. Salles, M.A.G. Vieira, A.I. Tavares, M.L.B. Carvalho, and O.S.F. Carvalho. Programação paralela em arquiteturas transputer. Technical Report 002/92, Departamento de Ciência da Computação - UFMG, 1992.

[Szw84] J. L. Szwarcfiter. *Grafos e algoritmos computacionais*. Editora Campus Ltda., 1984.

LDBA: Lenguaje Distribuido Basado en Actores

Francisco J. Mendieta
Alejandro Quintero
Departamento de Ingeniería de Sistemas y Computación
Universidad de los Andes
Apartado Aéreo 4976 - Bogotá - Colombia
e-mail: aquinter@andescol.bitnet
fmendiet@andescol.bitnet

Resumen: La investigación en la programación orientada por objetos ha mostrado la factibilidad de distribuir la computación utilizando el paso de mensajes de este paradigma y la posibilidad de obtener concurrencia a nivel de métodos dentro de un mismo objeto y en distintos objetos a la vez. El modelo actor de programación concurrente, presenta una alternativa de diseño para lenguajes distribuidos pues trae consigo el concepto de entidades autónomas -**actores**- cuya única forma de comunicación es el paso de mensajes. A continuación presentamos un lenguaje orientado por objetos de tipo C++ cuyos objetos activos -**actores**- se implantan transparentemente en procesadores distribuidos.

Palabras claves: Programación Orientada por Objetos, Programación Concurrente, Lenguajes Actores, Lenguajes para Sistemas Distribuidos, Sistemas Distribuidos

1. Introducción

El desarrollo de nuevas metodologías para la construcción de sistemas distribuidos y paralelos, ha llevado al desarrollo de nuevas áreas de investigación como la programación concurrente orientada por objetos y los lenguajes orientados por objeto como mecanismos para su especificación y descripción. Los sistemas son descritos entonces como un conjunto de objetos que interactúan entre si de manera concurrente mediante protocolos uniformes de comunicación. Las ventajas que ofrece el paradigma de programación concurrente orientado por objetos son las siguientes: a) los lenguajes orientados por objetos permiten expresar más explícitamente las relaciones que existen entre elementos del mundo real y sus contrapartes en el software, b) la comunicación por medio del paso de mensajes básica en los sistemas distribuidos, es reflejada en el paradigma del paso de mensajes usado por los lenguajes orientados por objetos [1].

El modelo actor de programación concurrente introducido por Carl Hewitt [2], permite modelar objetos compartidos con cambios locales en su estado, reconfigurar dinámicamente la conectividad entre objetos y el procesamiento concurrente inherente a sus objetos -**Actores**-. Más recientemente, Gul Agha [3] definió un modelo actor con un número pequeño de primitivas muy poderosas que son la base de la construcción del lenguaje ACT++ [1] y de nuestra implantación distribuida de este lenguaje llamada **LDBA** -Lenguaje Distribuido Basado en Actores- [4]. Este artículo describe el proyecto de implantación de LDBA en el cual intentamos verificar la utilidad de la programación de actores en lenguajes imperativos orientados por objetos, analizando las posibilidades de reutilización del software, y la factibilidad de construir jerarquías de herencia que nos permitirán lograr nuestro principal objetivo: la concurrencia.

La organización de este artículo es la siguiente: la sección 2 describe el lenguaje LDBA, la sección 3 presenta las conclusiones de nuestro proyecto de implantación del lenguaje y finalmente la sección 4 exhibe la bibliografía utilizada.

2. LDBA: Lenguaje Distribuido Basado en Actores

LDBA es una implantación distribuida del lenguaje ACT++ [1] el cual está basado en el modelo de Gul Agha [3]. En términos de implantación LDBA es una extesión de C++ [5] con una jerarquía de clases que permiten explotar el paralelismo intrínseco del modelo actor en una red local de máquinas Apollo (APOLLO COMPUTER INC.).

2.1. Primitivas de LDBA

El modelo actor es un modelo de computación concurrente en el cual la computación es llevada a cabo por actores que se comunican mediante el paso de mensajes. El modelo tiene cuatro elementos básicos: actores, colas de correo, comportamientos y relaciones.

Un **actor** es un objeto activo autocontenido. La interacción entre actores puede ocurrir solo a través del envío de mensajes. Cada actor está asociado a una única **cola de correo**, cuya dirección sirve como identificación del actor. Los mensajes enviados a un actor, son guardados en su cola de correo y son leídos uno a uno en estricto orden FIFO. Si la cola está vacía cuando el actor está listo para recibir el siguiente mensaje, el actor es bloqueado hasta que un mensaje llegue a la cola. Un actor A puede enviar un mensaje a un actor B, solo si A conoce la dirección de la cola de correo de B. Las direcciones de las colas de correo pueden ser enviadas como parte de un mensaje. Estas dos últimas propiedades permiten configurar dinámicamente la conectividad entre actores.

El **comportamiento** de un actor determina como reaccionar a los requerimientos especificados en el mensaje procesado. Un comportamiento o definición de clase actora , está definido por un cuerpo de código llamado **behavior script**. Este script contiene definiciones de métodos y **relaciones** o variables de instancia . Estas relaciones son los nombres de otros actores a los cuales el actor puede enviar mensajes -el modelo es homogéneo y las únicas entidades que pueden existir son actores-. Las relaciones del modelo de Agha son variables solo para lectura y no para escritura ya que la concurrencia inherente a nivel de instrucción no se puede garantizar si existe dependencia de datos entre instrucciones, pues cada sentencia es ejecutada concurrentemente -granuralidad fina- excepto cuando hay restricciones secuenciales por el orden casual. En LDBA las relaciones pueden ser modificadas gracias a la granuralidad gruesa de nuestra implantación -ejecución concurrente de métodos- haciendo el modelo más compatible con los lenguajes imperativos: de esta manera un actor puede recibir dinámicamente nombres de otros actores y cambiar mediante asignaciones sus variables de instancia.

En el procesamiento de un mensaje un actor puede hacer tres acciones: crear nuevos actores, enviar mensajes a otros actores y especificar un comportamiento de reemplazo. En el modelo de Gul Agha las dos primeras son opcionales y la tercera es obligatoria; sin embargo en LDBA el especificar un comportamiento de reemplazo solo debe hacerse cuando un actor ha modificado de alguna forma sus relaciones y quiere que este cambio en su estado sea permanente. A continuación detallaremos cada una de estas acciones.

2.1.1. La creación de actores

En LDBA podemos distinguir dos tipos de objetos: activos y pasivos. Los objetos activos son aquellos que de alguna forma son subclases de **Actor**, que es una clase definida en el lenguaje, mientras que los objetos pasivos constituyen todas las demás clases que el usuario defina. Los actores son creados dinámicamente en ejecución a medida que se necesitan mediante el operador **New** -no confundirlo con el new de C++-. Recibe como parámetro el nombre de la clase actora a crear, por ejemplo Mbox UnaMbox = New("UnaClaseActora") y retorna la cola de correo asociada al actor creado. Otros actores pueden enviar mensajes al nuevo actor usando esta cola de correo. Un ejemplo del encabezado de una clase actora sería:

```
class UnaClaseActora: public Actor {...};
```

2.1.2. El envío de mensajes

El invocar métodos de un objeto pasivo tiene la semántica de un llamado a una función y no lo discutiremos -podemos asociarlo con las transmisiones sincrónicas de ABCL/1 [6]-. La invocación de un método de un actor se hace mediante el paso de un mensaje asincrónico. El paso asincrónico de mensajes es soportado a través de objetos predefinidos como son las colas de correo, que también traen consigo la concurrencia inter-objeto: métodos de distintos objetos ejecutándose simultáneamente. Dos tipos de mensajes vamos a distinguir: **mensajes de requerimiento** y **mensajes de respuesta**.

Un mensaje de requerimiento es usado para invocar un método mientras un mensaje de respuesta es utilizado para entregar el resultado de la invocación de un método. Dos tipos de colas de correo están disponibles para soportar estos dos tipos de mensajes: las **Mboxes** y las **Cboxes**. Las Mboxes modelan las colas de correo de un actor mientras que las Cboxes permiten la programación con **futures** [7] que corresponden de alguna manera a una promesa de entrega: la computación es efectuada por un proceso paralelo que puede seguir activo después de que el future haya sido devuelto, los futures se confunden de esta manera con el valor de los datos a entregar.

Un mensaje de requerimiento consta del nombre del método a ser ejecutado por el actor receptor y los argumentos para el método. Si se quieren recibir respuestas de la invocación del método, se debe especificar una Cbox dentro de los argumentos como identificación del actor continuación [2], que es el actor que recibirá la respuesta del mensaje de requerimiento. Si no se

especifica el actor continuación tendremos asincronismo puro y si se especifica tendremos las transmisiones anticipadas de ABCL/1 [6]. Por ejemplo, si tenemos un actor **sumador** cuya Mbox es **mboxSum** y que posee un método **Sume** que nos permite sumar dos enteros, la invocación del método sería:

```
mboxSum << "sumador::Sume" << entero_1 << entero_2 << miCbox;
```

donde **miCbox** indica la Cbox donde se dejará el resultado de la suma.

Un actor puede referirse a su Mbox, mediante la variable predefinida **MiMbox**. Siguiendo nuestro ejemplo, el método **Sume** del actor sumador obtendría los parámetros de su invocación así:

```
MiMbox >> par_1 >> par_2 >> UnaCbox;
```

Como el nombre del método fue utilizado para seleccionar su invocación correspondiente, en MiMbox solo se reciben los parámetros. El número de parámetros en una invocación a un método puede ser fácilmente parametrizado, ya que los parámetros son en realidad enviados a un objeto **Correo** de las Mboxes el cual es retornado cuando la Mbox recibe el nombre del método a invocar.

Los mensajes de respuesta como CboxResp << suma, son colocados en las Cboxes que implantan la espera por necesidad. Un actor puede recibir de una Cbox mediante el operador >> y enviar a una Cbox con el operador <<. Si en una recepción hay un resultado para ser entregado al actor, le es entregado inmediatamente, de lo contrario el actor se bloquea hasta que un mensaje llegue. Un actor puede revisar si un mensaje de respuesta ha llegado a una Cbox usando la operación **in()** sobre la Cbox.

Las razones para la existencia de las Cboxes son principalmente el evadir el desdoblamiento de un actor que necesita una respuesta, en múltiples continuaciones que esperan por ella y el permitir una implantación eficiente de un actor que no puede proceder hasta que una respuesta sea recibida.

2.1.3. Especificación de un comportamiento de reemplazo

Un comportamiento es escrito usando las primitivas de actores y las construcciones imperativas de C++. Las instrucciones en un comportamiento son ejecutadas secuencialmente. La operación

llamada **become** permite cambiar el comportamiento de un actor dinámicamente en ejecución. Un actor puede ser asociado a distintos comportamientos en diferentes instantes del tiempo. Estos comportamientos podrían no tener los mismos métodos o estructuras de datos internas. La única entidad que no cambia en la vida de un actor, es la dirección de su cola de correo.

En el modelo de Agha, la operación become necesita un nombre de un comportamiento y una lista de relaciones -que puede ser muy larga si el actor está en lo profundo de una jerarquía de herencia-, haciendo que las estructuras de datos utilizadas por los actores sean pequeñas. En nuestro modelo, la operación become recibe un apuntador a un nuevo actor o la pseudo-variable **Self**, haciendo la estructura interna de los actores más flexible y compacta. La pseudo-variable Self, se utiliza para especificar como comportamiento de reemplazo al actor mismo.

La operación become juega dos papeles importantes en el modelo como son: es el único medio que tiene un actor para cambiar su estado, es decir, para hacer permanente cualquier cambio en sus relaciones y mantiene la consistencia de las relaciones del actor, ya que junto con los mensajes guardados en la cola de correo permite tratar el siguiente mensaje no procesado en forma serial.

En nuestra implantación, el usuario es el responsable de indicar si un actor es serializado -ejecuta un método a la vez- o no, especificando en el método constructor de la clase la constante **SERIALIZADO** como argumento para la construcción de la parte actora del objeto. Se sabe que un actor debe ser serializado, si en algún momento modifica sus relaciones y hace este cambio de estado permanente mediante una operación become. Por ejemplo, si sabemos que el actor **Almacén** al tratar el mensaje **retire**, decrementa su relación **cantidad** en el valor solicitado, su constructora seguiría el siguiente esquema:

```
Almacen::Almacen (void)
:Actor (SERIALIZADO)
{cantidad = MAX_ARTICULOS;}
```

y su método **retire** -teniendo en cuenta que hay artículos suficientes- sería:

```
void Almacen::retire (void)
{int pedido;
MiMbox >> pedido;
cantidad -= pedido;
become (Self);
}
```

Si no se especifica nada al construir la parte actora de Almacén -mediante :Actor(..)-, el actor Almacén sería por defecto **NO_SERIALIZADO**, es decir, permitiría el tratamiento simultáneo de métodos correspondientes al actor Almacén sin ningún mecanismo de sincronización, con la posibilidad de inconsistencias en el estado del actor cuando se hagan cambios en el valor de sus relaciones.

Un resultado significativo de la operacón become, es el potencial de concurrencia dentro del actor. La concurrencia intra-objeto -varios métodos de un objeto ejecutándose al mismo tiempo- naturalmente aparece cuando el comportamiento corriente continúa después de especificar su comportamiento de reemplazo.

2.2. Operaciones remotas

La existencia de las colas de requerimiento -Mboxes- y de las colas de respuesta -Cboxes- nos hace pensar en una implantación distribuida del modelo actor como la hace LDBA, que aprovecha el paso de mensajes entre actores para implantar tanto su creación como su ejecución remota de métodos, ambas en forma transparente para el usuario. Para lograr ésto, hemos definido un lenguaje muy sencillo con la misma filosofía del lenguaje de definición de interfaces que utiliza **RPC** para sus operaciones remotas. En este lenguaje se declaran para cada clase actora, los métodos de su interfaz que son requeridos para la computación. Con esta información y la ayuda de la herramienta **LDBA_parser** podemos generar el código necesario para la creación de actores por nombre -note que el argumento del New es una cadena de caracteres- y para la ejecución de los métodos de actores por procesos en máquinas remotas -que también son identificados por nombre-.

El archivo que describe los métodos de las clases actoras que se invocarán, presenta la siguiente forma:

```
methods <nombre_clase_actora> in <archivo_encabezado_clase> {
    void <nombre_metodo_1> ();
    ...
    void <nombre_metodo_n> ();
}
```

donde:

- nombre_clase_actora es el nombre de la clase actora cuyos métodos pueden ser accesados remotamente por medio de mensajes de requerimiento.

- archivo_encabezado_clase es el archivo de encabezado de la clase -.h- donde se definen de acuerdo a la sintaxis de C++ los métodos y los atributos de la clase actora.

- nombre_metodo_i es el nombre de un método de la interfaz del actor que se da a conocer para que pueda ser accesado por un mensaje de requerimiento.

La distribución de la computación es posible debido a la creación remota de actores y a la posibilidad de comunicarse entre ellos por medio de mensajes. La creación de actores en las diferentes máquinas del sistema se hace en forma balanceada por una autoridad central que decide la máquina de creación con base en el número de actores presentes en las máquinas en un instante determinado. Esta autoridad central se determina mediante la función **start()** que debe colocarse al principio del **main()** de la aplicación: está encargada de sincronizar los procesos remotos que intervendrán en el desarrollo del programa, asignando los papeles de autoridad central o servidor y de esclavos a las distintas máquinas del sistema. De igual forma para terminar la ejecución global del programa se debe invocar la función **end()** al final del mismo.

Finalmente en cada una de las máquinas del sistema, se ejecutan procesos de comunicación encargados de la creación de actores por nombre -New()-, el envío y la recepción de mensajes de respuesta -manejo de las Cboxes-, el envío de mensajes de requerimiento -ejecución de métodos por nombre- y la recepción de parámetros para los mismos -manejo de las Mboxes-, consultas a las Cboxes -in()-, y especificación de comportamientos de reemplazo -become()-.

3. Conclusiones

- En LDBA la operación become es usada la mayor parte del tiempo para la sincronización sobre un objeto compartido. Sinembargo puede resultar un caos en la estructura del programa puede, ya que un actor es libre de tener un comportamiento con muy poca o ninguna relación con el comportamiento actual. El uso indiscriminado del la operación become puede ser tan dañino como el uso indiscriminado del **goto**.

- La versión actual de LDBA, tiene aún restricciones que serán mejoradas con el tiempo; el desarrollo de una interfaz que permita la entrada-salida desde cualquier método en ejecución y un manejo de errores distribuidos apropiado, son las siguientes etapas del desarrollo. En el

futuro se considerará la tolerancia a fallas y se tendrá un modelo de control completamente distribuido.

- La implantación de la concurrencia intra-actor e inter-actor, hace que el uso de la creación de procesos pesados pueda degradar la eficiencia del sistema y en ocasiones limitar el tamaño de los problemas a solucionar. Sinembargo, la creación de estos procesos pesados trae consigo la uniformidad y la transportabilidad de la implantación actual a distintas máquinas con sistema operacional UNIX. En el futuro se trabajará en una implantación eficiente que utilice **threads**.

4. Bibliografía

[1] D. Kafura, "ACT++: Building a Concurrent C++ using Actors", Journal of Object-Oriented Programming, 1990.

[2] C. Hewitt, "Viewing Control Structures as Patterns of Passing Messages", Journal of Artificial Intelligence, Junio, 1977.

[3] G. Agha, "A Model of Concurrent Computation in Distributed Systems", MIT Press, Cambridge, MA, 1986.

[4] F. J. Mendieta, "LDBA: Lenguaje Distribuído Basado en Actores", Tesis de Grado, Universidad de los Andes, SantaFe de Bogotá, 1991.

[5] B. Stroustrup, "The C++ Programming Language", Addison-Wesley, Menlo-Park, CA, 1986.

[6] A. Yonezawa, Shibayama, "Modelling and Programming in an Object-Oriented Concurrent Language ABCL/1", Object-Oriented Concurrent Programming, MIT Press, Cambridge, MA, 1987.

[7] H. Lieberman, "Concurrent Object-Oriented Programming in ACT 1", Object-Oriented Concurrent Programming, MIT Press, Cambridge, MA, 1987.

Categorical Models for Relevant Logics

M. Claudia Meré*
Dept. Informatica
Pontifícia Universidade Católica do Rio de Janeiro
R. Marquês de São Vicente 225, 22453 RJ, Brazil
E-mail: mazzuca@inf.puc-rio.br

Valéria V. C. de Paiva
Computer Laboratory
University of Cambridge
New Museum Site, CB2 3QG Cambridge, England
E-mail: vcvp@cl.cam.ac.uk

Abstract

In this note we try to understand and present some of the relevant logics in what we think is the mathematically tidier way. Since we want to use Category Theory for our models, we want to have (cut-free) Gentzen-style sequent calculus formulations for our systems. We present the systems, discuss their semantics (categorical and algebraic) and then relate them to the existing Relevant Logics.

KEYWORDS: Gentzen sequent calculus, Linear Logic, Relevant Logics, Category Theory, Categorical Logic.

1 Introduction

Intuitionistic Propositional Logic when originally presented by Gentzen in a sequent-calculus formulation **LJ** has three structural rules Permutation, Contraction and Weakening:

$$(P)\frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, B, A, \Gamma' \vdash C} \qquad (C)\frac{\Gamma, A, A, \Delta \vdash C}{\Gamma, A, \Delta \vdash C} \qquad (W)\frac{\Gamma, \Delta \vdash C}{\Gamma, A, \Delta \vdash C}$$

*This paper was written when the author was visiting the Computer Laboratory, University of Cambridge.

There has been recently a reasonable amount of work on the hierarchy of logics that result when we get rid of some (or all) of the structural rules. The resulting logical systems have been called *substructural logics*. The lack of the rule of Contraction means that each occurrence of a formula counts as a different formula, and so the contexts on the left of the turnstile must correspond to multisets and not only to sets. This accounts for the name of *occurrence logics*, and they have the particularity that the complexity of a sequent must increase in the course of a proof. Also, the lack of the Weakening rule shows that the logic is *non-monotonic* in the weak sense that if we are able to derive a formula C from a multiset Γ of premisses we can say nothing about the same C being derivable from Γ plus other premisses. It also means that in the antecedent of the sequent we must only put assumptions that are important (relevant) to the derivation of a formula.

Girard's Linear Logic gets rid of the two structural rules of Contraction and Weakening in one go (neither they hold nor they are derivable rules). As a result we obtain the well-known splitting of the logical conjunction and disjunction into two versions: one multiplicative and one additive. Also, the (multiplicative) implication we get behaves 'like' a relevant implication.

In this note we try to understand and present some of the relevant logics in what we think is the mathematically tidier way. Since we want to use Category Theory for our models, we want to have (cut-free) Gentzen-style sequent calculus formulations for our systems. We present the systems, discuss their semantics (categorical and algebraic) and then relate them to the existing Relevant Logics.

2 Tensor-Implication Logic

As it is usual in relevantists' work, we will begin our development by presenting the system that deals with the fragment implicational of the logic and we will add the other connectives later. But, because we want to work with Category Theory we believe that it is easier to begin with the fragment which has the multiplicative conjunction, or tensor $\otimes$ (and its respective unit), and the implication $\multimap$ connectives. We present a Gentzen-style sequent calculus and call this system **TI** :

Structural Rules:

$$(Identity)\frac{}{A \vdash A} \qquad (P)\frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, B, A, \Gamma' \vdash C} \qquad (Cut)\frac{\Gamma \vdash A \quad A, \Delta \vdash B}{\Gamma, \Delta \vdash B}$$

Logical Rules:

$$I_l\frac{\Gamma \vdash A}{\Gamma, I \vdash A} \qquad I_r\frac{}{\vdash I}$$

$$\multimap_l \frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C} \qquad \multimap_r \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}$$

$$\otimes_l\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \qquad \otimes_r\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}$$

Due to the lack of traditional rules of Contraction and Weakening, in this calculus capital Greek letters (eg. Γ) stand for (finite) multisets (or bags) of assumptions. It might help the reader to think of the multiplicative conjunction or tensor connective $\otimes$ as a *juxtaposition* of data, so one can reorder data but one cannot duplicate or erase it. Tensor is a weak form of conjunction and the constant I is the identity for it. The linear implication $A \multimap B$ can be thought of a function which looks at its argument exactly once.

This calculus admits cut elimination [1], i.e. every derivation which uses the rule (*Cut*) can be transformed into a cut free derivation. This has very pleasant consequences: the consistency of the system and the subformula property (a cut free proof contains only subformulas of the sequent it proves). The system **TI** above is called in some papers with computational linguistic motivations the Lambek Calculus with Permutation **LP** ([2]).

The lack of Weakening means that in the antecedent of the sequent we must only put assumptions that are important (relevant) to derive a formula. Hence, we think of **TI** as one of the simplest Relevant Logics[1]. The basic idea of Relevant Logic is to constraint odd entailments and so, a well formed implicational formula is one where the antecedent really produce the consequent.

The system **TI** corresponds to the implicational plus cotenability (and the constant called t) fragment of the relevant logic **RW** ([4],[5]).

There are several different notations for algebraic models of **TI**. We follow the algebraic approach of thinking first of tensor logic and then adding implication. Thus, to model tensor logic we use any *ordered monoid* [6], the idea is that the partial order $\leq$ models entailment and the monoidal operation $\circ$ models both the comma and the tensor connective. To model tensor-implication logic we use any *closed poset* [6].

Categorical Logic is a well-established branch of Mathematical Logic and we take it for granted in this work. As every poset is a trivial kind of category (the category where all the maps from A to B are the "same" map) it makes sense to

[1] as a relevant logic it is a paraconsistent logic too [3].

look for more general models which preserve the distinction of maps. This is an oversimplified description but one can say that the natural generalization of the concept of an ordered symmetric monoid is a *symmetric monoidal category* [7], and the generalization of a closed poset is a *symmetric monoidal closed category* [7].

3 Extensions of the Tensor-Implication Calculus

When we are working with a sequent-calculus with all the structural rules, like **LJ**, we have that these structural rules hold in the sense up-to-down and then, with the help of the other rules, it is possible to derive the other direction. But, when any of the structural rule lacks we are not more able to conclude all the senses down-to-up from the permitted rules. So, if we want only to drop a structural rule of the calculus but preserving the whole effect of the other rules, we must add whichever of the other senses of the rules is lacking. Thus, for example, if we drop the rule of Weakening it is not more possible to conclude that the converse of the Contraction rule is valid, and this is important when we are worried with the number of times a formula appears in a derivation.

Therefore, we have that the system **TI** plus Contraction$\downarrow$ is the same as the implicational fragment of Relevant Logic **R** with cotenability and the constant t ([4],[5]). And, on the other hand, if we think that the converse of the Contraction (called *premise repetition*) is also allowed we obtain another traditional relevance logic, $\mathbf{RM0}_{\rightarrow}$ ([4], [5]). The system $\mathbf{RM0}_{\rightarrow}$ has the implication axioms of the relevant logic plus the mingle axiom. Meyer has shown that **RM** (the system obtained adding the mingle axiom to the full system **R**) is not a conservative extension of

RM0$_{_}$.

We have that the algebraic structure needed to model **TI** plus Contraction($\downarrow$) is any closed poset $(M, \leq, \circ, e, \multimap)$ where $\circ$ is *square-increasing*, i.e., $a \leq a \circ a$ for all a in M. We call this structure a *Church monoid.*

If the monoidal operation $\circ$ is also *square-decreasing*, i.e. $a \circ a \leq a$ then, $\circ$ is *idempotent* and we have a model of **TI** plus Contraction.

As in the previous section we want to generalise the ideas of the algebraic models. The categorical generalization of a Church monoid is is a *symmetric monoidal category with diagonals* [12]. And, on the other hand if we want to model the converse of the Contraction rule we need the 'converse' notion of having diagonals. This corresponds to the concept of *monoid* [7]. So, a symmetric monoidal closed category **C** with diagonals where each object is a monoid models **TI** plus Contraction.

If we add to the system **TI** the rules corresponding to the additives & and $\oplus$ (with their units 1 and 0 respectively) we obtain the system corresponding to the Intuitionistic Linear Logic **ILL**. The algebraic models are the closed posets with finite meets and joints and the categorical structures are the symmetric monoidal closed categories with finite products and coproducts (called *linear categories*).

If we add to **TI** plus Contraction an axiom corresponding to the distributivity of the additives & and $\oplus$, i.e.the scheme axiom $\Gamma, A \,\&\, (B \oplus C) \vdash \Gamma, (A \,\&\, B) \oplus C$, we obtain the corresponding system to the relevant logic **R** plus distributivity (with cotenability). We can model it easily with distributive structures.

Also, if we add to **ILL** the rules of Contraction and Weakening we obtain the Intuitionistic Logic system **LJ** ([13]) because with these two structural rules we can

prove that the connectives $\otimes$ and $\&$ are the same. We can model it through *cartesian closed categories with coproducts* [14].

Finally, we want to describe a semantical way of adding the structural rules of Contraction to Intuitionistic Linear Logic to get a relevant logic. This idea has been independently described by Prof. Kosta Dosen [15] but in that paper semantics is not presented. By contrast here the (categorical) semantics is the main motivation because the model so obtained is very elegant but now we will have in the calculus the **S4**-like rules that are difficult to deal with.

The idea is simple enough. We start with a sequent presentation of **ILL** and we add a modality, which we call $\Box$. This $\Box$ modality satisfies the **S4**-like rules ([16]) as well as the Contraction rule. We obtain that: the sequents provable in the calculus correspond to the sequents provable in the relevant logic **R** (without distributivity of the additives); and we obtain easily categorical models for this calculus.

4 Conclusions

A lot of work needs to be done. First as we would like to understand the whole family of Relevant Logics systems like E and T need to be analyzed. Also we need to develop term assigment systems and λ-calculus for all the logics discused here and to study the models like Kripke.

Finally, we want to make connections with several areas of Computer Science more explicit, mainly with some formal methods to model concurrency.

References

[1] J.Y. Girard and Y. Lafont. Linear Logic and lazy computation. In *Tapsoft 87*, LNCS 250, Springer-Verlag, pages 52–66, 1987.

[2] J. van Benthem. The Lambek Calculus. In *Categorical Grammars and Natural Language Structures.* R. Oehrle, E. Bach and D. Wheeler, eds., 1988.

[3] A.I. Arruda A survey of Paraconsistent Logic. In *Mathematical Logic in Latin America.* A.I. Arruda, R. Chuaqui and N.C.A. da Costa, eds. North-Holland Publishing Company, 1980.

[4] A.R. Anderson and N.D. Belnap. *Entailment, The Logic of Relevance and Necessity.* Princeton University Press, Princeton, 1975.

[5] J.M. Dunn. Relevance Logic and entailment. In *Handbook of Philosophical Logic, Vol III*, chapter III.3. D. Gabbay and F. Guenthner, eds. Reidel Publishing Company, 1986.

[6] M. Hyland and V. de Paiva. Lineales. Technical report, University of Cambridge, 1991.

[7] S. Mac Lane. *Categories for the Working Mathematician.* Springer-Verlag, New York, 1971.

[8] A. Avron. The semantics and proof theory of Linear Logic. *Theoretical Computer Science*, (57):161–184, 1988.

[9] G.E. Minc. Cut-elimination theorem for Relevant Logics. *Journal of Soviet Mathematic*, (6):422–428, 1976.

[10] V. de Paiva. A dialectica model of the Lambek Calculus. Technical report, University of Cambridge, 1991.

[11] R. Meyer and R. Routley. Algebraic analysis of entailment i. *Logique et Analyse*, 15:407–428, 1972.

[12] M.E. Szabo. *Studies in Logic, Vol 88, Algebra of Proofs.* North-Holland, Amsterdam, 1978.

[13] G. Takeuti. *Proof Theory.* North-Holland, Amsterdam, 1975.

[14] J. Lambek and P.J. Scott. *Introduction to Higher Order Categorical Logic.* Cambridge University Press, Cambridge, 1986.

[15] K. Dosen. Modal translations in susbtructural logics. *The Journal of Philosophical Logic (to appear).*

[16] D. Prawitz. *Natural Deduction, A Proof-Theoretical Study.* Almqvisty & Wiksell, Stockholm, 1965.

A DYNAMIC TESTER FOR USE WITH OSI NETWORKS

Nasser Modiri, M.Sc., D.Phil., A.M.I.E.E., M.I.E.E.E.

EleTek Systems
1107 Second Ave, Suite 704
Redwood City, CA 94063, USA.

Abstract.

The widespread acceptance of Open Systems Interconnection (OSI) by most large computer manufacturers and procurement agencies means that many computer networks are now being installed that are based on a standard protocol defined by the International Standards Organisation (ISO). Although prior to the commissioning of the communication subsystem within a networked computer the protocols making up the subsystem are subjected to rigourous conformance tests, should a fault be suspected in an operational system it is also necessary to provide a facility to allow the operation of a system to be tested dynamically via the network. Such instruments are referred to as dynamic testers and they are particularly useful during the commissioning of a new network, for example, and also if a system connected to an operational network is suspected of being faulty.

1 INTRODUCTION

The dynamic tester to be described has been designed to operate as an intelligent client system for the dynamic testing of a network server such as a large file or mail server. It is a flexible tool that can be used to interactively perform specific conformance tests.

Essentially, associated with each test, there is a precompiled set of frames which can be transmitted to a system under test via the network. The response is then compared with the expected response.

The dynamic tester has many features similar to a conventional conformance tester [1,2,3,4,5,6] except that it is designed to operate over an operational network rather than, say, in a conformance laboratory. Also, it is designed to test a complete protocol suite rather than, say, an isolated protocol layer.

The paper comprises five sections. After this section, Section 2 gives the rationale behind the work, Section 3 describes the overall structure of the dynamic tester in which the test definition and run-time components are described, Section 4 describes some implementation aspects of the dynamic tester, and finally, Section 5 concludes the paper.

2 RATIONALE

Prior to acceptance of a system to be connected to an OSI network, it is likely that the system would have been subjected to rigourous conformance tests. It is, however, possible that some faults may only manifest themselves in an operational situation. There is also the possibility of an untested system being connected to the network.

The approach taken in designing the dynamic tester has been to provide a user-friendly interface (all operations are menu driven) and to minimise the level of expertise required by the operator. The selected protocols to be used in the OSI Environment (OSIE) - that is, the configuration profile - will have been previously defined, and hence the conformance requirements for a system connected to the network will be known. The conformance tests to be carried out, therefore, aim to establish that a system adheres firmly to the standards selected. The dynamic tester is designed to operate as a client which can be used to test a remote server.

It performs this function by transmitting previously compiled frames to the remote system (server) under test. Each frame is designed to test a specific step in an overall transaction involving the server; for example, the establishment of an association between, say, the File Transfer Access and Management (FTAM)[7] application entity in a file server and the tester. Associated with each test there is a frame which has previously been compiled representing the expected response.

This frame is then compared with the actual response frame received from the remote server. If the frames are the same, the test is assumed to be successful; if they are different, however, then the tester will identify the field(s) that is (are) in error, the layer to which it relates, and the actual symbolic name of the field as used in the appropriate standards document. In order to test all aspects of a particular server, a number of test scenarios are devised. Frames to be transmitted and frames expected to be received, relating to each test, are created prior to delivery of the tester. Since there may be many tests and the amount of data relating to a complete test suite might be large, the data relating to each test are first stored on disc. Subsequent analysis may then take place in an off-line mode rather than an on-line mode.

3 OVERALL STRUCTURE

The overall structure of the software associated with the dynamic tester is comprised of two main components the Test Definition component and the Run-time component.

Essentially, the Test Definition component is used to configure the tester prior to delivery whilst the Run-time component, as its name implies, is used to perform the actual dynamic test operations over, an operational network in the field. The functional overview of the dynamic tester is shown in Fig. 1.

A schematic diagram showing the structure of the hierarchical menu of each component is shown in Fig. 2. As can be seen from the figure, the Test Definition component comprises three activities - *Configure*, *Scenario Entry* and *Precompile*. Once these activities have been carried out, there are further utilities which are used to generate the Run-time version of the tester. Using this procedure it is possible to make available to a user only the Run-time version of the tester which has been specifically tailored to the OSI being monitored and managed.

3.1 Test Definition

The Test Definition software is used to configure a tester for use in a particular OSIE prior to delivery of the tester. It is comprised of three major components - *Configure*, *Scenario Entry* and *Precompile*. Each is now described separately.

Configure

The *Configure* component which is run first, is used to select the particular configuration profile that is being used with the OSIE. It is comprised of two main parts, namely *Profile Selection* and *Generate*. An outline of their functions is presented here.

The set of protocols to be used with the dynamic tester is first selected from a list of possible protocols using the *Profile Selection* component. An example of its use is shown in Fig. 3 (a..h). Essentially, the operator first selects the appropriate protocol suite needed to make up a tester for use in this environment, then, as each layer in the reference model is selected, all the protocol standards supported for the selected layer are listed. The user is thus able to select the appropriate protocol to be used for each layer simply by using a selection button on the mouse. Then as each protocol is selected, this information is added to the configuration file of the tester prior to compilation.

The operator can compile the configuration selected by selecting the *Compile* option as shown in Fig. 4. Hence, after compilation, just the object code necessary for the selected protocols of the *Precompile*, *Compile* and *Analysis* components is produced.

Once the set of protocols to be used in the environment has been selected, it is then necessary to generate a precise definition of the Protocol Data Units (PDUs) associated with each selected protocol. Many of the ISO standards have a number of optional features associated with them, for example with respect to the inclusion of a particular field within a PDU. Hence, having selected a configuration profile, it is then necessary for the user to select the inclusion (or otherwise) of those fields that are optional in each protocol making up the selected profile. This is accomplished using the interactive software tool, *Generate*, an example of which is shown in Fig. 5 (a..b). Essentially, as each protocol layer is reselected, then the list of PDUs associated with the previously selected protocol is presented. The user then selects each PDU in turn and, if any of the fields are optional, the user is prompted to specify whether the field should be included or not. Thus after this phase a precise template for each protocol in the suite will have been generated. These are then used in the subsequent *Precompile* phase and later during the *Compile* and *Analysis* phases of the Run-time component.

Scenario Entry

The overall test of an operational system is comprised of a number of sub-tests each of which is known as a scenario.

Typically, associated with each scenario is a sequence of previously compiled frames - together with their expected responses - each of which has been selected to test a specific aspect of the server. Defining scenarios for a test case of a System Under Test (SUT) (that is, a real open system) can be arrived at by looking at the conformance requirements relating to the ISO standards being used. As an example, assume it is required to devise a sequence of tests for a server based on the FTAM application entity. A typical test scenario that has been devised is shown in Table 1. The scenario shows the design to test the association establishment and termination components of FTAM. The layer PDUs making up the sent and expected frames associated with each test in a scenario are kept in a table known as the test definition table.

The *Scenario Entry* component, therefore, has been devised to allow an operator to interactively enter the contents of this table. As can be seen, this is a two dimensional table the first column of which specifies the name of the frames involved in each test. The second column denotes the direction of the frame (that is, sent (S) or expected (E)). The remaining eight columns then indicate the PDUs relating to each layer that make up the composite frames that will be generated for the test case. It is then the contents of the test definition tables that are passed to the *Precompile* component.

Precompile

The *Precompile* phase involves first looking up the PDUs that have been entered into the test definition tables to ascertain the composition of each sent and expected frame then, for each PDU, assigning the fields to either a default value or a value specified by the operator. The operator is first requested to select the application entity - FTAM or EM[8] - to be tested and a scenario number specifying the selected test. The PDUs in each test frame relating to the selected test are then listed and the operator is able to enter the appropriate value for each field in each PDU. As an example, assume it is required wish to precompile a test suite relating to an FTAM server. A typical sequence of events is as shown in Fig. 6.

First the user in part (a) selects FTAM and the test scenario number 001. In response, the user is then presented with the contents of the test definition table relating to this test scenario. This is shown in part (b) and is the same as that shown earlier in the *Scenario Entry* section. The test sequence is thus concerned with the establishment and termination of an FTAM association. Before this can be carried out, however, a transport connection between the tester and the server must first be established, the first two entries - sent and expected - performing this function. With the aid of the mouse the user is then able to proceed to define the fields in each PDU making up each frame and as an example, the windows in part (c) illustrate how the fields relating to the Transport Connection Request (CR) PDU (TPDU) [9] component of the first frame are presented. In practice these are then followed by the fields in the Unit Data Request (DT) Network PDU (NPDU) [10], the Un-

numbered Information Request (UI) Logical Link PDU (LPDU) [11], and the Data Request (DT) Medium Access Control (MAC) PDU [12]. As can be seen, the user is presented with a default value for each field and this can either be accepted (by pressing the return key on the keyboard) or a new value specified.

This procedure continues for each frame in the selected test scenario and, for each frame, a data file is produced containing the corresponding octet string. Only those fields that contain non-site-specific data values can be entered in this way but in practice these are the majority and only a small number of values - for example, the fully qualified address of the server under test - need be entered at run time.

An important component of the *Precompile* software is the set of so-called Precompile Protocol Engines (PPEs). Essentially there is a separate PPE for each protocol layer and its function is to produce the precompiled octet string for each layer PDU. The sent and expected frames for each test are thus a collection of the outputs from these PPEs.

A subdirectory is assigned to all the functionality required from a protocol layer and a layer PPE handles the management of the operations of the layer. In this way, development of the software can be devised in a straight forward way and more importantly, since the ISO standards are evolving, changing or adding new standards to the existing standards considered for the dynamic tester can be carried out more readily. Further, debugging of the dynamic tester components can be completed more quickly by running the PPEs independently from one another. In the *Precompile* stage for each PDU making up a subset frame, the PPEs display the elements of the PDU. Against each element typical default values are displayed and the user is then given the option of either accepting this value or entering a new value. After all the fields of a PDU have been entered, the PPE software for that layer produces the corresponding octet string. In addition to converting each field, the layer PPE also performs operations such as the computation of the number of octets in the PDU - the length indication - and the computation of the checksum for the precompiled octets.

Whenever possible maximum information is made available to the operator, so that he/she does not have to refer back to the standard documents, for example, to verify the encoding rules which define a PDU. Hence, the operator is able to use the dynamic tester to devise valid PDUs and consequently test scenarios with a minimum knowledge of the standards.

3.2 Run-time components

The Run-time component is comprised of four parts, as shown in Fig. 2. These are - *Compile*, *Concatenate*, *Test*, and *Analysis*. Each of these components now will be described with more details about the *Test* component given as a part of the Implementation Aspects of the dynamic tester.

Compile

The main function of the *Compile* stage is to allow a user to interactively specify any site-specific fields associated with the PDUs that make up each frame in the test scenarios. The input to the *Compile* stage is the set of data files that were generated at the *Precompile* phase - prior to delivery of the tester - and also the set of test definition tables that were entered during the *Scenario Entry* phase. The *Compile* component is selected by the user from the main menu shown in Fig. 2. The sequence of operations with the *Compile* component are similar to those of the *Precompile* component described earlier except that here the user is asked to enter only the site-specific fields associated with each PDU in the set of test frames.

As an example, consider the case where a user wishes to test an FTAM server. After selecting the *Compile* option, the user would then select FTAM followed by a scenario number. This is shown in part (a) of Fig. 7.

The first PDU of this test is the CR TPDU and, as can be seen in part (b) of this figure, the user is presented with the precompiled (that is, site-independent) fields and is prompted with the first site-specific element - that is, Source Reference. The user then either accepts the default value displayed or specifies another value. This procedure continues until all the fields in each PDU relating to this scenario have been defined - that is, all the PDUs in both the sent and expected frames. A second example is shown in Fig. 8 part (a..d).

This figure relates to a PDU of the FTAM entity itself and, as can be seen, fields are specified in an Abstract Syntax Notation One (ASN.1)[13,14] form. In the example the user is being prompted for the concrete value notation of the FTAM initialize request (INIRQ) PDU.

In the absence of a protocol stack in the dynamic tester - the octet strings created are transmitted directly onto the transmission medium - the user must know the fully qualified network address of the FTAM server under test. The user then enters each subaddress of the network address in the appropriate layer PDU as it is being compiled.

The defined fields making up each PDU are then processed by the appropriate layer Compile Protocol Engine (CPE). There is a separate CPE for each protocol layer and its function is to produce the corresponding octet string for the PDU. Thus the Transport Layer CPE will compute the length indication field and the checksum fields prior to compilation.

Similarly, for those protocol layers that are defined in ASN.1, the user will specify each field in this form and the corresponding CPE then compiles these into their concrete syntax form. Each compiled PDU is then stored in a separate file. The compiled PDUs making up each sent and expected frame must then be concatenated together to produce the octet strings that are transmitted and received.

Concatenate

Generally, a scenario comprises a number of sent and corresponding expected frames. These frames are made by concatenating the PDUs produced during the *Compile* phase.

Thus for the scenario test 001 of FTAM, eight data files would be produced containing a sequence of sent and expected frames as originally defined in the test definition table for this scenario. This *Compile - Concatenate* cycle is repeated for each scenario. The dynamic tester is then ready to start the actual test cycle.

Test

After a user has produced the compiled and concatenated set of frames relating to each scenario, the server may now be dynamically tested via the network. Clearly, there are two approaches that may be adopted: either each frame in a scenario is transmitted separately, or, the complete suite of frames relating to a scenario is transmitted contiguously. In practice, since the tests are to be carried out on an operational system, then the first approach may result in the server timing-out between tests. To overcome this problem, when a scenario is invoked, each frame is transmitted and a simple comparison - in terms of length and content - is made to determine whether the fields in the response frame are valid prior to sending the next frame in the scenario.

In addition, however, as each response frame is received is stored in a data file in the event of a more detailed analysis being required during the subsequent *Analysis* phase. Should a response frame be deemed to be incorrect, then the scenario is aborted and the *Analysis* phase is entered to determine the error.

In the event of the latter, however, then the association and network connections are both closed.

Analysis

After the *Test* phase, the frames received from the SUT in response to each transmitted frame will be stored in a data file. If all the test in each scenario have been carried out successfully, then all is well and there is no need to proceed further. If a test has failed, however, then the actual received frames can be analysed to ascertain the fault by entering the *Analysis* phase.

To use the *Analysis* component the user selects *Analysis* option from the main menu together with the name of the server - FTAM or EM. In response the user is presented with a summary of the previous *Test* phase. An example is shown in Fig. 9. As can be seen, first a summary of the success or failure of each scenario is presented and, in the event of a scenario failure, a summary is also presented of the success or otherwise of each test frame in the scenario. Thus in the example scenario 003 was aborted because frame 002 was invalid, Table 2 shows the frames and PDUs that comprise the 003 test of FTAM.

The user must now determine which field(s) and in which PDU(s) are invalid. To help the user, the contents of the failed frame are automatically passed and a summary is presented in a third window. In addition, the contents of the Protocol Control Information (PCI) of the invalid PDU are presented in a fourth window and those fields that are different from those expected are highlighted. Clearly, if the parsing operation fails, then only those fields that have been successfully parsed are presented.

4 DEVELOPMENT ENVIRONMENT AND TESTING

As has been described, the dynamic tester is not designed to test a single protocol layer associated with communication subsystem but rather to dynamically test complete protocol stack including a selected application entity; for example, a file server that is based on FTAM. The dynamic tester is also based on a Sun Workstation® [15] and all software is written in C [16] running under the UNIX™ operating system. The physical interface between the Sun and the network will, of course, be network dependent. Two boards for use with a Technical Office Protocol (TOP)[17] and Manufacturing Automation Protocol (MAP)[18] network. These are the MVME 374 Ethernet™ Controller board and the MVME 372 advanced MAP Controller board respectively.

A schematic diagram showing the actual test system is given in Fig. 10 part (a). The interface board in the tester does not contain the full protocol software. Rather it simply contains software to transmit the compiled frames associated with a test scenario onto the transmission medium and, as each response frame is received, to pass the response frames to the host processor for analysis/verification.

The development of the tester was carried out in a number of steps. First precompiled frames and their associated precompiled responses were used. The responses also has deliberately introduced errors associated with them and the software was able to detect an identify those fields that were in error and also any incorrect sequence errors.

Secondly, the FTAM server associated with the ISODE [19] software running on top of Transmission Control Protocol (TCP) was used to test a real system.

A schematic is shown in Fig. 10 part (b). As can be seen, the ISODE software comprises only layer 4 upwards but this formed a suitable vehicle for carrying out tests on the FTAM layer entity in particular.

The actual tester software occupies 1.8 Mbytes of main memory to test a TOP (or MAP) FTAM-based file server.

5 CONCLUSIONS.

Conventional conformance test tools [20,21] are extremely complex and often take a considerable amount of time to set up. In contrast the dynamic tester described in this paper has been designed to perform a reasonably detailed level of tests on a system that is connected to an operational network. The aim is for the tester to be used during the commissioning phase of a new network and also as a part of the general instrumentation available to a network manager of an OSI network.

6 ACKNOWLEDGEMENTS.

The work reported in this paper has been partially funded by The Networking Centre Ltd. of Hemel Hempstead, England. The author would like to acknowledge this support and also the many useful discussions with them about the facility offered by the tester.

7 REFERENCES.

[1] BICC Data Networks, "ISOLAN Test Unit", BICC Data Networks, 1987.
[2] VANCE, "ATS 1000, Network Analysis and Test Systems", VANCE Systems Inc.
[3] National Computing Centre, "Transport Layer Tester", COMMS-AID, The Networking Centre, 1987.
[4] RETIX, "RETIX NETBIOS Test/Validation Tool", RETIX The International Company, 1989.
[5] RETIX, "RETIX Session Test/Validation Tool", RETIX The International Company, 1989.
[6] RETIX, "RETIX Transport Test/Validation Tool", RETIX The International Company, 1989.
[7] ISO 8571/1-4 - 1988 FTAM Service/Protocol Specification.
[8] CCITT - 1988, CCITT: Message Handling Systems and Service Overview, International Telegraph and Telephone Consultative Committee Recommendation. X.400.
[9] ISO 8072/3 - 1987 Transport Service/Protocol/Specification
[10] ISO 8348/2 - 1987 Network Service/Protocol/Specification
[11] ISO 8802.2 - 1987 Logic Link Control Service/Protocol/Specification
[12] IEEE - 1985 Token Bus Access Method IEEE 802.4.
[13] ISO 8824 - 1987, OSI: Specification of Abstract Syntax Notation One (ASN.1).
[14] ISO 8825 (1987), OSI: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).
[15] Sun-3 Architecture, "A Sun Technical Report", Sun Microsystems, Inc. 1986.
[16] B. W. Kernighan and D. M. Ritchie, "The C Programming Language", Prentice-Hall, 1987.
[17] S. A. Farowich, "Communicating in the technical office", IEEE Spectrum, pp. 63-67, Apr. 1986.
[18] M. A. Kaminski, "Protocols for communicating in the factory", IEEE Spectrum, pp. 56-62, Apr. 1986.
[19] T. R. Marshall, "The ISO Development Environment", Vol. 1-4, The Wollongong Group, 1988.
[20] R. Matthews, K. Muralidhar and S. Sparks, "MAP 2.1 Conformance Testing Tools", IEEE Trans.On Software Eng., Vol. 14, No 3, pp.363-374, Mar. 1988.
[21] R. Matthews, K. Muralidhar and M. K. Schumacher, "Conformance Testing: Operational aspects, tools and experiences", Proc. 3rd Int. Workshop Protocol Specification, Testing and Verification. Amsterdam, The Netherlands: North-Holland, 1986.

UNIX™ is a registered trademark of AT & T and Bell Laboratories.
Sun Workstation® is a registered trademark of Sun Microsystems Inc.
Ethernet™ is a registered trademark of the Xerox Corporation.

Frame Name	S/E	FPDU	APDU	PPDU	SPDU	TPDU	NPDU	LPDU	MPDU
Connection req	S					CR	DT	UI	DT
Connection conf	E					CC	DT	UI	DT
Initialize req	S	INIRQ	AARQ	CP	CN	DT	DT	UI	DT
Initialize resp	E	INIRP	AARE	CPA	AC	DT	DT	UI	DT
Terminate req	S	TERRQ	RLRQ	RS	FN	DT	DT	UI	DT
Terminate resp	E	TERRP	RLRE	RSA	DN	DT	DT	UI	DT
Disconnect req	S					DR	DT	UI	DT
Disconnect conf	E					DC	DT	UI	DT

Table 1: *FTAM 001 Test Definition table.*

Frame Name	S/E	FPDU	APDU	PPDU	SPDU	TPDU	NPDU	LPDU	MPDU
Connection req	S					CR	DT	UI	DT
Connection conf	E					CC	DT	UI	DT
Initialize req	S	INIRQ	AARQ	CP	CN	DT	DT	UI	DT
Initialize resp	E	INIRP	AARE	CPA	AC	DT	DT	UI	DT
Select req	S	SELRQ		DT	DT	DT	DT	UI	DT
Select resp	E	SELRP		DT	DT	DT	DT	UI	DT
Deselect req	S	DESRQ		DT	DT	DT	DT	UI	DT
Deselect resp	E	DESRP		DT	DT	DT	DT	UI	DT
Terminate req	S	TERRQ	RLRQ	RS	FN	DT	DT	UI	DT
Terminate resp	E	TERRP	RLRE	RSA	DN	DT	DT	UI	DT
Disconnect req	S					DR	DT	UI	DT
Disconnect conf	E					DC	DT	UI	DT

Table 2: *FTAM 003 Test Definition table.*

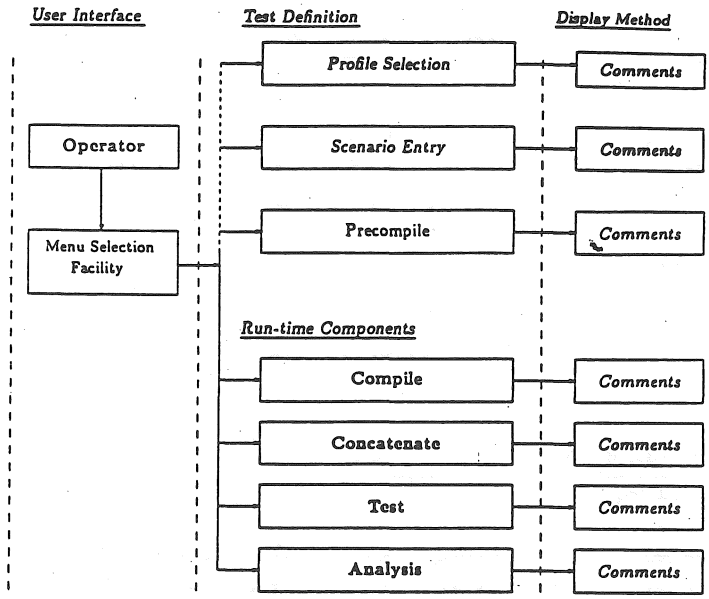


Figure 1: *Functional Overview of the Dynamic Tester.*

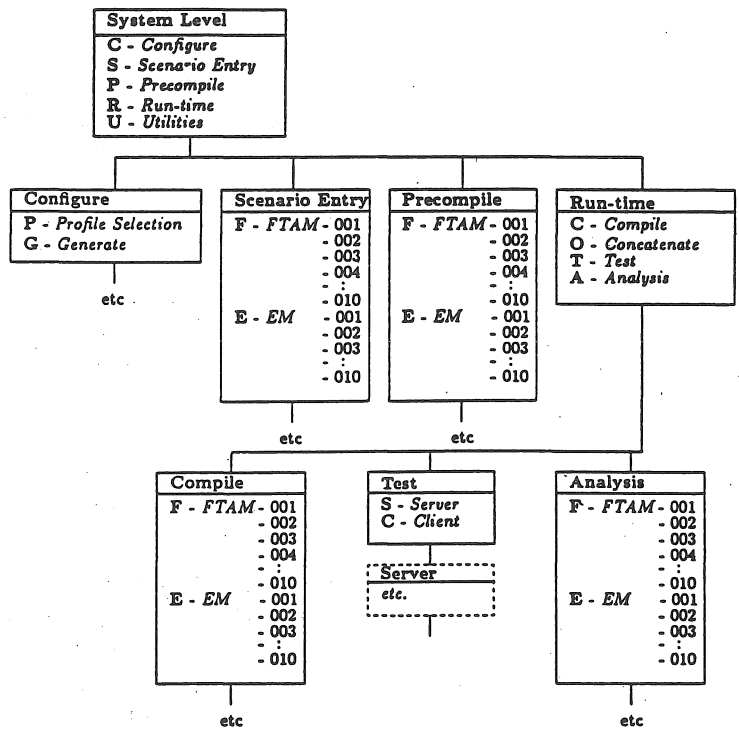


Figure 2: *Hierarchical menu of the Dynamic Tester.*

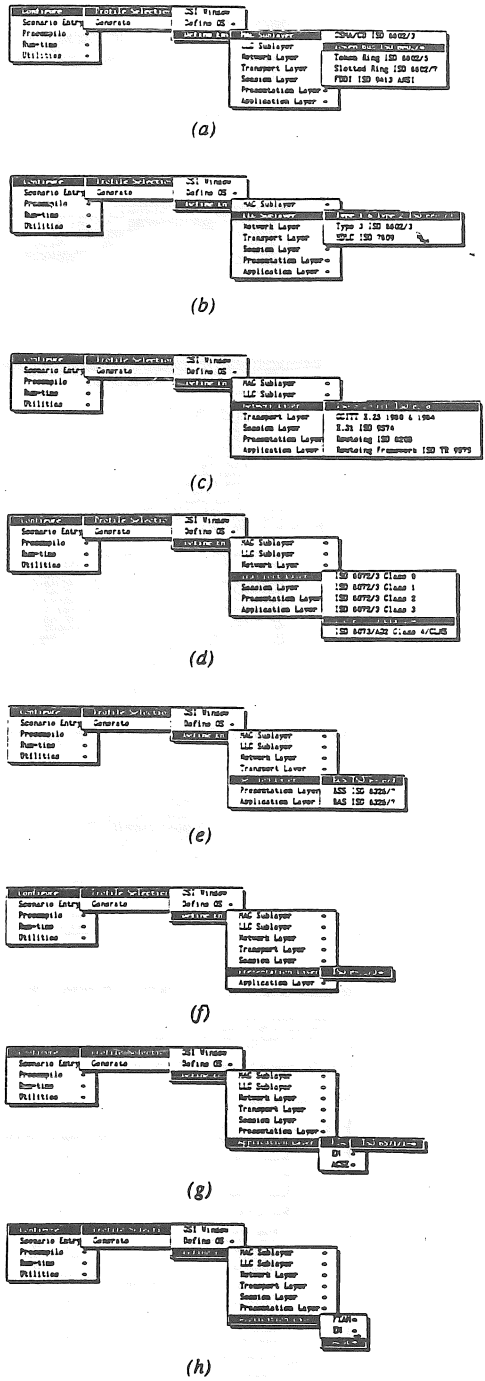
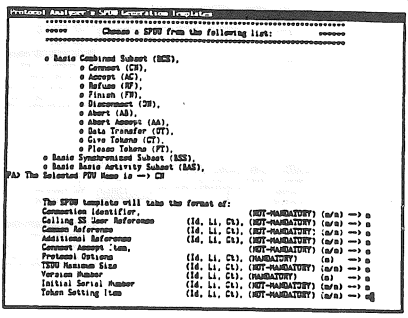


Figure 3: *Profile Selection: (a) MAC Sublayer; (b) LLC Sublayer; (c) Network Layer; (d) Transport Layer; (e) Session Layer; (f) Presentation Layer; (g) ACSE Application Entity; (h) FTAM Application Entity.*

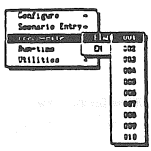
Figure 4: *Compile the selected profile.*

(a)

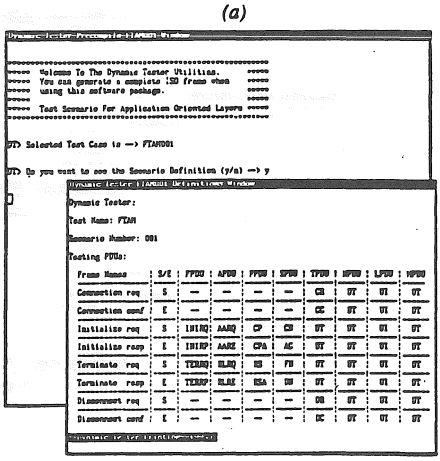


(b)

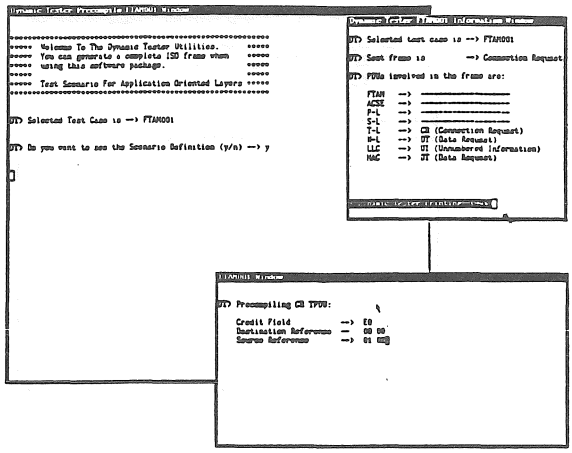
Figure 5: *Generate phase: (a) select Session Layer; (b) generate CN SPDU template.*



(a)

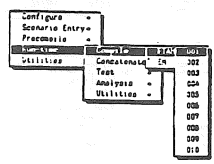


(b)

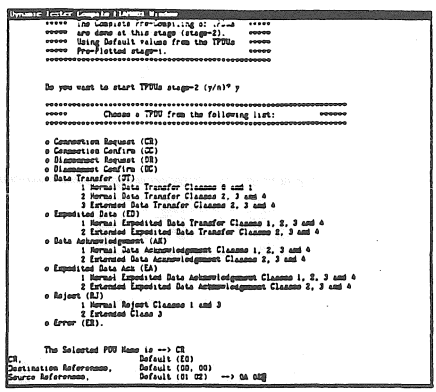


(c)

Figure 6: Precompile phase: (a) Select FTAM 001 scenario; (b) Test Definition table; (c) defining all the CR TPDU fields.

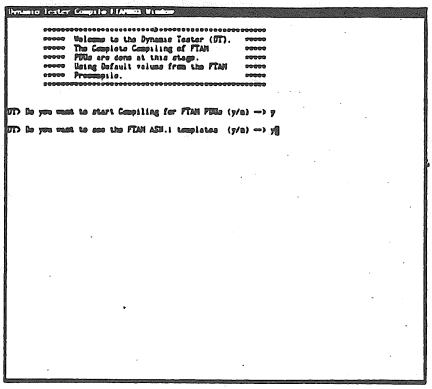


(a)

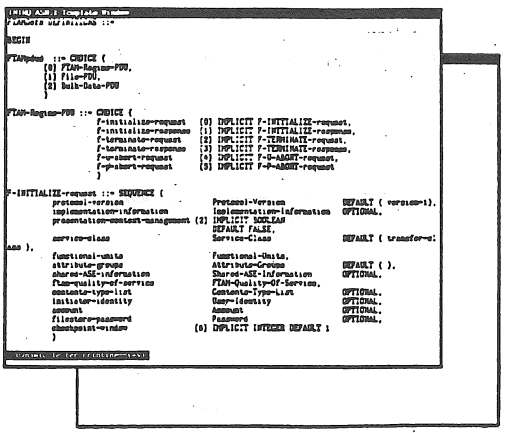


(b)

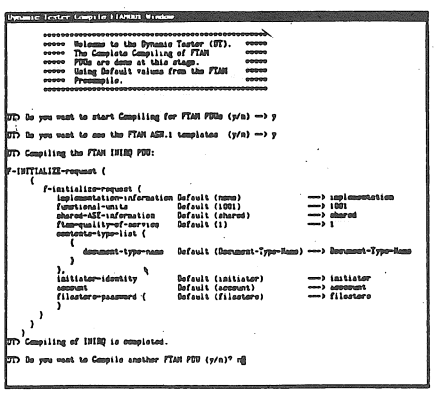
Figure 7: Compile phase: (a) Select FTAM 001 Scenario; (b) Define site specific fields of the CR TPDU.


Dynamic Tester Compile FTAM001 Window
Welcome to the Dynamic Tester (DT).
The Complete Compiling of FTAM
PDUs are done at this stage.
Using Default values from the FTAM
Precompile.
DT> Do you want to start Compiling for FTAM PDUs (y/n) --> y
DT> Do you want to see the FTAM ASN.1 templates (y/n) --> y

(a)


FTAMpdus ::= CHOICE {
[0] FTAM-Regime-PDU,
[1] File-PDU,
[2] Bulk-Data-PDU
}
FTAM-Regime-PDU ::= CHOICE {
f-initialize-request [0] IMPLICIT F-INITIALIZE-request,
f-initialize-response [1] IMPLICIT F-INITIALIZE-response,
f-terminate-request [2] IMPLICIT F-TERMINATE-request,
f-terminate-response [3] IMPLICIT F-TERMINATE-response,
f-u-abort-request [4] IMPLICIT F-U-ABORT-request,
f-p-abort-request [5] IMPLICIT F-P-ABORT-request
}
F-INITIALIZE-request ::= SEQUENCE {
protocol-version Protocol-Version DEFAULT { version-1},
implementation-information Implementation-Information OPTIONAL,
presentation-context-management [2] IMPLICIT BOOLEAN
DEFAULT FALSE,
service-class Service-Class DEFAULT { transfer-c
ass },
functional-units Functional-Units,
attribute-groups Attribute-Groups DEFAULT { },
shared-ASE-information Shared-ASE-information OPTIONAL,
ftam-quality-of-service FTAM-Quality-Of-Service,
contents-type-list Contents-Type-List OPTIONAL,
initiator-identity User-Identity OPTIONAL,
account Account OPTIONAL,
filestore-password Password OPTIONAL,
checkpoint-window [0] IMPLICIT INTEGER DEFAULT 1
}

(b)


Dynamic Tester Compile FTAM001 Window
Welcome to the Dynamic Tester (DT).
The Complete Compiling of FTAM
PDUs are done at this stage.
Using Default values from the FTAM
Precompile.
DT> Do you want to start Compiling for FTAM PDUs (y/n) --> y
DT> Do you want to see the FTAM ASN.1 templates (y/n) --> y
DT> Compiling the FTAM INIRQ PDU:
F-INITIALIZE-request {
f-initialize-request {
implementation-information Default (name) --> implementation
functional-units Default (1001) --> 1001
shared-ASE-information Default (shared) --> shared
ftam-quality-of-service Default (1) --> 1
contents-type-list {
{
document-type-name Default (Document-Type-Name) --> Document-Type-Name
}
},
initiator-identity Default (initiator) --> initiator
account Default (account) --> account
filestore-password { Default (filestore) --> filestore
}
}
}
}
DT> Compiling of INIRQ is completed.
DT> Do you want to Compile another FTAM PDU (y/n)? n

(c)

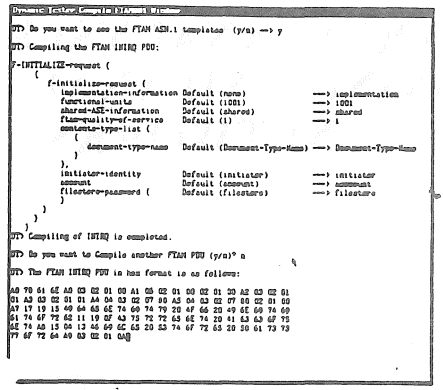


Figure 8: Compile phase: (a) Compiling FTAM INIRQ PDU; (b) FTAM INIRQ PDU definition; (c) Define site-specific fields; (d) Concrete syntax of INIRQ PDU.

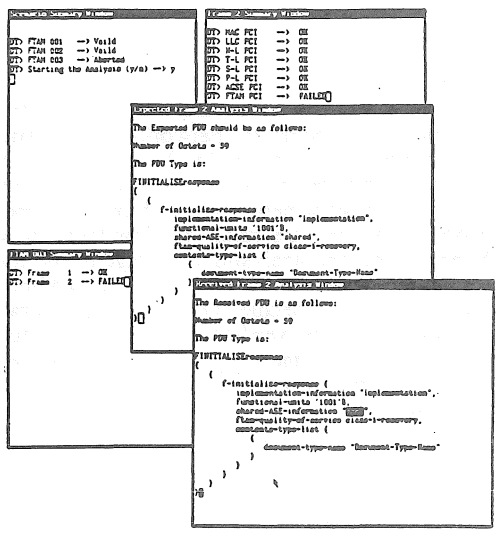


Figure 9: Analysis result of scenario FTAM 003.

A PROTOCOL ANALYSER FOR THE MONITORING AND ANALYSIS OF OSI NETWORKS

Nasser Modiri, M.Sc., D.Phil., A.M.I.E.E., M.I.E.E.E.

EleTek Systems
1107 Second Ave, Suite 704
Redwood City, CA 94063, USA.

Abstract.

The widespread acceptance world-wide of Open Systems Interconnection (OSI) has resulted in an increasing number of computer networks being installed that are based on internationally agreed standard protocols defined by the International Standards Organisation (ISO). Associated with any computer network there must be instrumentation for the monitoring and analysis of the network performance. Most instruments currently available for use with OSI networks, however, are relatively low-level in so much that they are concerned mainly with monitoring and analysing the usage of the transmission medium. In contrast, the protocol analyser described in this paper, in addition to performing such functions, is also capable of performing detailed analysis of the interactions between any two systems connected to a network and the verification of their operation against the adopted protocol standards of the network. Also, the results of the analysis operations are presented in a high-level, graphical style that requires a minimum of knowledge about communication protocols on the part of the user.

1 INTRODUCTION

The acceptance worldwide of Open Systems Interconnection (OSI) by most large computer manufacturers and procurement agencies means that many computer networks are now being installed that are based on a standard protocol defined by the International Standards Organisation (ISO). Although prior to the commissioning of the communication subsystem within a networked computer the protocols making up the subsystem are subjected to rigourous conformance tests, should a fault be suspected in an operational system it is also necessary to provide facilities to allow the operation of a system to be monitored and any interactions anaylsed to ensure it is functioning correctly. This is particularly important during the commissioning of a new network, for example, and also when a system is being added to an operational network. This paper is concerned specifically with this aspect of OSI networks and describes the results of a research project that has been concerned with the design and implementation of a network protocol analyser that has been implemented for this purpose.

Most network analysers [1,2,3,4,5,6,7] that are currently available, simply log all network traffic - frame transmissions - in a passive way and then subsequently use the logged data to determine such statistics as the distribution of traffic levels around a network, the level of usage of each network segment, time and load related problems, verification of network error handling, and so on. Normally, a profile of the performance of the network is created and the network behaviour characteristics are then presented through a series of summary screens. The network characteristics are generally presented as bar, histogram or performance - against - time sampled graphs. Hence from the analysis, one can deduce, for example, network utilisation as a line chart with the actual data points for each sample over the sampling period. This type of statistic can be very useful to a network manager since, from the captured data, nodes, for example, that generate large quantities of data can be singled out for further analysis.

In addition to such features, some analysers provide some rudimentary facilities for investigating the actual frame interactions between two selected systems. Such analysers, therefore, are referred to as (network) protocol analysers. Normally, however, with most current protocol analysers the gathered data is presented in a basic ASCII-hex form and hence such analysers generally need expert operators who have a thorough knowledge of the protocols being analysed. In contrast, the protocol analyser to be described, provides a user-friendly facility that reduces the level of expertise required from the operator when analysing the behaviour of (OSI) networks that are based on ISO.

The paper comprises seven sections. After the introduction, Section 2 gives the rationale behind the work and the approach adopted. Section 3 describes the overall structure of some of the software tools developed. Section 4 gives some detail about how to configure and generate an analyser for a specific network environment. This is then followed in Section 5 by a description of the run-time components of a prototype analyser, and in particular, how the user can control interactively both the analysis and style of presentation of the gathered data. In the ISO standards documents, the Protocol Data Units (PDUs) exchanged between two peer protocols are in a precisely defined format and Section 5 describes how the messages gathered are decoded prior to analysis with examples of the analysis and presentation procedure adopted. Finally, Section 6 discusses some of the implementation details of the analyser in a more general context and Section 7 concludes the paper.

2 RATIONALE

The adoption of a layered architecture for the ISO reference model for OSI [8,9,10,11,12] means that each protocol layer can be treated as a separate entity for analysis and test purposes. Each layer in the ISO model provides a well defined set of user services to the layer above it and operates according to a precisely specified protocol.

Moreover, the structure (format) and contents of the PDUs exchanged between two correspondent (peer) protocol entities are also precisely defined. Typically, a PDU created by one layer will comprise its own Protocol Control Information (PCI) together with the PDU(s) from the higher layer(s) in its user data field. This is shown in diagrammatic form in Fig. 1.

The effect of this is that by analysing the composite information in each frame transmitted between two systems over a network, it is possible to deduce:

- the identity (address) of the sender and intended recipient of each frame;
- the PDUs making up the frame;
- the protocol layer to which each PDU relates;
- the protocol control information associated with each layer.

As will be expanded upon later, each frame transmitted over an OSI network consists of an integral number of octets - 8 bit bytes - and most network analysers simply present the PCI relating to each layer as a string of ASCII - hex characters. To deduce the cause of a suspected fault with such analysers, therefore, requires considerable expertise and knowledge on the part of the user.

Within the standards documents relating to the ISO protocols well understood techniques have been adopted for defining both the user services provided by a protocol layer and also the specification of the operation of a protocol entity; the former is in the form of a time sequence diagram whilst the latter is normally in the form of an event-state table.

Clearly, therefore, if when analysing the operation of either a single protocol layer or a complete communication subsystem the various PDUs and associated events can be related directly to the relevant graphical template, then it will be more straight forward for a user of the analyser firstly, to follow (and hence understand) the sequence of events which have occurred in relation to the standards document and secondly, to determine the cause of any errors should these arise.

When carrying out a particular test run, the amount of data gathered can be considerable. Thus, when presenting the results of a test run it is important that this is done in an interactive way so that the user can firstly, proceed with the analysis operation at his or her own rate and secondly, dictate the sequence and amount of detail which is to be displayed. For example, if a user is interested only in, say, the Transport Layer [13,14], then it is important to display only the messages which relate to it. Alternatively, if a user is interested in the behaviour of a complete communication subsystem, then the interactions and PDUs relating to all the layers must be presented. In either case, however, the presentation of this information should be in a top-down manner so that the level of detail displayed is under the control of the user. This is the approach which has been adopted for the design of the analyser to be described.

3 OVERALL STRUCTURE

Before describing the detail features of the prototype network protocol analyser that has been developed [15,16], it is perhaps helpful to first identify the major software components that make up the analyser. A schematic of the various components is shown in Fig. 2.

Essentially, there are two major software components - one concerned with Environment Selection and the other Run-time components. The function of the Environment Selection component is to allow the user to tailor the analyser to the requirements of a particular open system environment. This is a one-off operation and, once carried out, the analyser is ready to be connected to the network.

The Run-time component is itself comprised of three sections - *Monitoring*, *Message Decode* and *Analysis*. The *Monitoring* component is concerned with the capture of messages from the network medium. Clearly this operates in real-time and the captured messages are then stored on disc to await further processing. The *Message Decode* component operates on the captured data and is responsible for parsing each logged message to identify the individual PDUs that make up the message. Once this has been carried out, the *Analysis* software is then run. The latter provides the user with an interactive facility to allow the style of presentation of the analysed results to be selected. Each of the major components will now be described.

4 ENVIRONMENT SELECTION

It should be remembered, there is not just a single protocol standard for each layer in the ISO reference model. Rather there are a number of standards each offering a different level of functionality within the context of the layer to which it relates. Moreover, as experience is gained from the use of a particular standard, so revisions are made to it, often with significant differences from the original version(s). Clearly, therefore, it is important when creating an open system environment for the administrative authority to specify not only the particular standards to be used at each layer but also the version numbers of these standards. The resulting set of standards is then referred to as the configuration profile or road map of the environment. When selecting a network protocol analyser, therefore, it is necessary to ensure that it has the correct configuration profile for the network environment that is to be used. An example, of an environment that is based on the ISO protocols is MAP [17]. There is now a fully defined configuration profile for this known as MAP3.0 and indeed it is for this environment that the network protocol analyser to be described has been initially targeted.

4.1 Configuration profile

The set of protocols to be used with the analyser are selected from a list of possible protocols using an interactive software tool known as *Configure*. An example of its use is shown in Fig. 2 (a..h).

The example relates to the selection of a configuration profile for MAP3.0. The operator first selects the appropriate protocol suite needed to make up an analyser for use in this environment. It should be noted that although the selections in the example are specific to the MAP3.0 analyser, the software is designed to be able to handle any combination of the standards specified in the menu. As can be seen, as each layer in the reference model is selected then all the protocol standards supported by the analyser for the selected layer are listed. The user is thus able to select the appropriate protocol to be used for each layer simply by using a selection button on the mouse. Then as each protocol is selected, this information is added to the configuration file of the analyser. The latter is included with the parsing code for each layer prior to compilation. Hence, after compilation, just the code necessary to parse the selected protocols will be produced.

4.2 PDU template generation

Once the set of protocols to be used in the environment have been selected, it is then necessary to generate a precise definition of the PDUs associated with each selected protocol.

Many of the ISO standards have a number of optional features associated with them; for example, with respect to the inclusion of a particular field within a PDU. Hence, having selected a configuration profile, it is then necessary for the user to select the inclusion (or otherwise) of those fields that are optional in each protocol making up the selected profile. This is accomplished using the interactive software tool *Generate* and an example of its use is shown in Fig. 4 (a..d).

As can be seen, as each protocol layer is selected, the list of PDUs associated with the selected protocol are listed. The user then selects each PDU in turn and, if any of the fields are optional, the user is prompted to specify whether the field should be included or not. Thus after this phase a precise template for each protocol in the suite will have been generated and this is used in the Run-time phase by the corresponding parser code generated as a result of the earlier configuration operation.

5 RUN-TIME COMPONENTS

Before describing the features of each of the Run-time components in detail, it is perhaps helpful first to outline a typical user dialogue with the implemented system. A schematic representing the overall menu is shown in Fig. 5.

As with all interactive systems, the menu is hierarchical in structure with the user selecting a specific option at each level in the hierarchy. Thus the highest (system) level menu allows the user to enter the required operational mode: *Monitoring*, *Message Decode* or *Analysis*.

Normally, the *Monitoring* mode is selected first as this is concerned with the logging of the frames transmitted on the transmission medium. The user will then select the *Message Decode* mode and this will use the previously created layer parsing code (engine), together with the PDU templates where appropriate, to deduce the individual PDUs PCI making up each logged message frame. Finally, the user selects the *Analysis* mode to present the results of the decoded message (frame) sequence to confirm or otherwise the correct operation of the targeted system.

5.1 Monitoring

The *Monitoring* mode allows the messages exchanged during a dialogue between two (or more) selected systems to be monitored and logged. The user first enters the (network) addresses (or names) of the two systems and these are first checked against a file - the system directory - that contains a list of all the names and corresponding addresses that are in use. The network addresses (or names) are then used as a filter to log only those messages which are exchanged between these two systems. Fig. 6 part (a) shows the operator has selected File Transfer Access and Management (FTAM)[12] as the application entity of interest and part (b) shows the operator entering the network addresses of the two nodes of interest.

The operator is then requested to specify the number of messages that it requires to capture during the *Monitoring* phase. Once the selected number of messages have been captured and stored on disk (or a timeout interval expires), the network protocol analyser is then ready to analyse them. This is achieved; firstly by parsing each message to ascertain the PDUs it contains - *Message Decode* - and then analysing the content and sequence of the PDUs to verify the correct operation of each system - *Analysis*.

5.2 Message Decode

The parsing code for each protocol in the selected configuration (produced by the *Configure* program) is known as a parser-engine. Also, associated with each parser-engine, there is a script file that manages the output of the parser-engine. There is then an additional script file which invokes each parser-engine in the correct sequence and also manages any data produced by them.

Essentially, as each parser-engine is called, it takes in a Presentation Stream (PS) of hexadecimal digits - the contents of a logged frame - parses it, and then displays the parsed contents in a readily understood form. Alternatively, if an error is detected when parsing the PS, then the operator is warned and a message indicating where and which octet(s) of the PS is(are) in fault is written against the faulty octet(s). Although all the PCIs relating to each protocol are parsed, only the essential fields in the PCI are printed. In addition, however, it is possible to request a more complete version of the PCI should this be required.

As an example of the *Message Decode* phase, assume that it is required to analyse the behaviour of two communication systems that are exchanging messages over a factory-wide broadband MAP network. A typical sequence of messages exchanged (in ASCII-hex form) with such a network are as shown in Fig. 7.

The message sequence relates to two FTAM protocol entities and these have been obtained by simply monitoring all transmissions over the factory-wide broadband transmission medium and using the physical network addresses of the two systems as a filter.

5.3 Analysis

As was mentioned earlier, because the amount of information gathered during a monitored period may be large, it is important to provide a facility to enable the user to interactively select, in a top-down way, the level of detail relating to the interactions which he/she is interested in. Thus, as can be seen in the menu shown earlier in Fig. 5, when the *Analysis* mode is selected, the user is asked to select the level of interactions which are of interest - System Level, Layer Level or Logical Sequence Checker (LSC). An example of each will now be presented. Each uses the sequence of messages shown earlier in Fig. 7.

System Level

The seven-layers in the ISO reference model can be divided into two groups according to their function - application-oriented (which embraces layers 5-7) and network-oriented (which embraces layers 1-4). Thus, when the System Level option is selected, the user is then asked to select the required functional group. As an example, Fig. 12 part (a) shows the user selecting the application-oriented layers and the interactions shown in part (b) and part (c) of this figure relate to the application-oriented group.

Essentially, the decoded message sequence is first preprocessed and only those messages which have PDUs relating to the application-oriented layers are analysed. As can be seen, the interactions are shown against a graphical template and the user steps through the interactions which have taken place (deduced by analysing the messages exchanged) simply by pressing a key on the mouse. Also, the lower analysis window acts as a commentary window and gives useful comments about the analysis process being carried out within each parsed frame. In addition, a vertical bubble bar is provided on the side of this window, the height of the bar represents the position of the displayed frame within all the messages captured.

A similar procedure is followed to display the interactions relating to the network-oriented group and an example sequence is shown in Fig. 13 part (a) and part (b).

A similar preprocessing phase is carried out but this time only messages which contain PDUs relating to the application-oriented layers analysed. Also, as can be seen, a different graphical template is used but, apart from this, exactly the same procedure is followed.

Clearly, if during the displayed sequence a fault is suspected, the user will then require additional or more detailed information to try to ascertain the cause of the fault. Normally, this will be related to a specific protocol layer and hence the Layer Level would then be selected with the suspected faulty layer as the target. In this way, the user is able to obtain first a high level view of the operation of the system then, should a fault be suspected, a more detailed view of the suspected area.

Layer Level

To analyse the interactions involving a specific layer, the user selects the Layer Level option and, as an example of its use, part (a) of Fig. 14 shows the user selecting the Layer Level option and part (b) shows the type of graphical template used.

The layer (protocol entity) selected is FTAM and the steps shown again relate to the message sequence in Fig. 7. For this type of analysis the messages are again preprocessed and only those which have PDUs relating to the selected layer are analysed. The style of presentation is intended to show the services provided by the layer and hence the graphical template used is a time sequence diagram.

As with the System Level display, the user also may select additional information if this is required in the lower analysis window. A scrollbar is also provided on the side of the window so that the user can go back to the previous PDU being displayed. Also, information is provided in the window to indicate which frame in a sequence is under analysis. Thus, in the example shown in part (c) of Fig. 14, the user has selected to display more information relating to the INIRQ FTAM PDU. As was described earlier, the various FTAM PDUs are defined in ASN.1 form and hence, since this definition is required (for each PDU type) for the decoding process outlined earlier, the various fields associated with the PDU are displayed against the corresponding ASN.1 template to aid readability. Clearly, if a complete sequence of messages is being analysed, in addition to analysing - and displaying - selected messages, it is possible to relate the sequence to the formal definition of the protocol entity of the layer under investigation. By analysing a sequence of frames, it is possible to ascertain the correct (or otherwise) functioning of the protocol behaviour.

The analyser must respond to any type of combination of PDUs which have been transmitted between the two nodes under consideration. For this reason, all the graphical templates devised can react dynamically to any combination of decoded PDUs. In particular, when preparing the graphical templates care has been taken to make them suitable to respond to the dynamic behaviour of a test run. The display interactions may start at any position in a test. Also, should a frame be encountered that is faulty (that is, has not been decoded), then the user is alerted of this.

As was depicted earlier in Fig. 2, each message contains PCI relating to a number of protocol layers and hence the first step is to decode each message into its constituent parts based on the known structure of each PDU as defined in the relevant standards document. The decoded PDU boundaries relating to the messages shown in Fig. 7 are thus as shown in Fig. 8 part (a) and part (b).

As can be seen, the first two messages comprise only PDUs up to the Transport Layer whilst the others comprise PDUs for the full seven-layers. In practice, this is readily determined since, after each parser-engine has been called - starting at the MAC Sublayer - a test is made to determine whether there is any data remaining in the frame buffers; if there is the next parser-engine is called, otherwise the parsing is complete.

To illustrate the above procedure, consider the first message block in the sequence. The various steps involved are shown in Fig. 9. First the protocol control information relating to the MAC Sublayer is determined by using the MAC parser-engine derived from the ISO 8802.4 standard. Similarly, the PCI relating to the LLC Sublayer is determined (again using the appropriate parser-engine derived from the LLC ISO 8802.2 standard document) and so on until the complete message has been parsed.

In the standards documents the structure and contents of each PDU up to and including the Session Layer [19,20] is defined in a fixed bit/octet string form. The PDUs associated with the Presentation Layer protocol machine [21,22]and the various application layer entities, however, are defined in an Abstract Syntax Notation (ASN.1)[23,24,25,26]. As an example, a PDU relating to the FTAM protocol is shown in part (a) of Fig. 10 and part (b..e) outlines how the related PDU is encoded.

As can be seen, each field (data element) in the PDU is of an abstract data type and the corresponding value making up the PDU is then encoded as:

- Type - this indicates the type of the data element; both simple types such as INTEGER, BOOLEAN etc. are used and also constructed types such as SEQUENCE (similar to a Record type) and SET;
- Length - this indicates the number of octets which follow in the contents field;
- Contents - this is the actual value of the data element, represented in a fixed (concrete) syntax form if it is a simple type, or a further type field if it is a constructed type.

Each PDU will have a different encoding (and hence decoding) procedure associated with it but, since the first octet always defines the type of the PDU, then the appropriate procedure can be selected and used to interpret (decode) the values associated with each data element. As an example, Fig. 11 shows the output of the parser-engines for the application support layers including the Session Layer.

Once the monitored and logged message sequence has been decoded - first into individual PDUs and then the various fields (elements) in each PDU identified - it can then be analysed and displayed to enable the user to observe the interactions that have taken place. Basically, in this phase a long file is generated relating to all the message frames parsed by the *Message Decode* and the output of the parser includes enough information in the generated file to determine if a PDU is in error.

When a PDU is detected to be in error, an audible and a visible signal are given to the user in the form of sounding the keyboard bell and flashing the screen, plus putting the erroneous PDU name in inversed screen mode.

A PDU in error is determined either by its Unique Identifier (UI) not being recognised as being a PDU belonging to the layer or while parsing the individual fields within a PDU an out of range field is detected.

There is also a help facility provided for searching through any of the parsed (decoded) files produced by the parser-engines in their appropriate windows. These windows may be invoked individually or in any combination. The contents of the parsed file, therefore, may be printed and used to identify any errors that may be present prior to selecting the *Analysis* mode. Alternatively, if there are a large number of frames being decoded, the parsed file may be large and a more user-friendly analysis option is thus offered to the user.

Operational modes

There are two alternative *Message Decode* processes and hence operational modes - Manual and Automatic. With the Manual mode of operation, the user has to specify the frame number to be decoded. Consequently, this has a slower performance rate, but a number of help facilities are provided at each stage of parsing the individual PDUs making up the frame. For example,the user can enquire about the parsed PDUs for the same layer in other frames in the sequence by simply paging the appropriate window. If at any time an error is encountered within a PDU being parsed the audible and visible bells are invoked. The biggest advantage of the Manual mode is that the parsing of frames does not have to be done in the same order as the stored sequence of frames. This mode is intended, therefore, as a means of examining the contents of individual frames rather than the sequencing of frames.

With the Automatic mode of operation, the user may elect to have the total parsing operation (that is, of all the logged frames) done automatically without the display of any of the parsed output. In addition, the user may elect to specify, say, a specific layer protocol in which case only the parsed output relating to that layer will be produced. In either event, however, the output of the Automatic mode process forms the input of the *Analysis* process.

Logical Sequence Checker (LSC)

A second type of analysis that can be done on the parsed PDUs is semantic analysis of the PDUs. This involves checking the sequence of PDUs and hence only when the parser-engines have syntactically checked the contents of all the frames in a logged sequence this is carried out.

The checking process involves a thorough knowledge of the interrelationships of the PDUs for a layer. This knowledge is based on the specification of the state-machine of the selected protocol of the layer under examination. All the possible valid sequences of PDUs relating to the operation of a protocol machine are stored in a table. This information is then used by the LSC to determine with a good probability, whether the sequence of the PDUs exchanged between two communicating protocol entities is in a valid sequence.

As an example, the contents of the sequence table for the FTAM protocol entity is shown in Fig. 15. This has been constructed by analysing the operation of the FTAM state-machine given in the ISO/DIS 8571/4 FTAM specification Document: '*' indicates a valid sequence and a blank an invalid sequence. As can be see, a protocol such as FTAM has 41 PDUs associated with it and hence manual sequence checking is difficult. The LSC thus takes much of the burdon away from the operator.

The selectable options available with the LSC provides the operator with the capability of checking for the correct sequence of the PDUs in the logged parsed messages for all the protocol entities supported by the analyser in any of the seven layers. The input file used by the LSC is again the output file produced by the *Message Decode* process. An example of the output of the LSC is shown in Fig. 16.

This again relates to the logged message (frame) sequence shown earlier in Fig. 7. As can be seen, the selected protocol layer under test is transport and, as each TPDU in the sequence is processed, an indication is given wether the PDU is valid or not (i.e., is or is not in the correct sequence). The algorithm of the LSC is based on a two dimentional table driven procedure. The incoming PDU under test is then used as one index and the last PDU processed as the other index.

6 DEVELOPMENT ENVIRONMENT AND TESTING

The network protocol analyser described has been implemented on a Sun Workstation® [27]. The programming language used is C [28] running under the UNIX™ operating system. The source code of the basic analyser - comprising the code necessary to analyse the MAC through Presentation Layers - occupies 1 Mbyte. Additional code is then required for each application entity supported; for example, FTAM requires 80 Kbytes and MMS [29,30] 100 Kbytes. A typical executable version of the analyser requires in the order of 3 Mbytes of main memory which includes the additional graphics and window software tools for the man-machine interface discussed.

The menu facility used with the analyser operates in the Suntools environment. SunView (Sun Visual/Integrated Environment for Workstations) graphic primitives are used for devising the interactive graphic analysis of the analyser. Only the SunView graphic package has been used to devise the graphics utilities. Hence, if there is a requirement for changing the graphic primitives to that of another package, changes can be made easily. There are three graphic programs associated with the analyser: *Schematic*, *Network Statistics* and *Analysis*. Since they all use the Sunview graphic primitives, they have been merged into a single program. An argument is then passed to the executable merged program, so that individual programs may be called independently from one another.

Testing

The physical interface to the analyser is, of course, network dependent. Essentially, however, it is through a communication card containing just the appropriate physical interface circuitry for the network being used. The filtering and message storage operations are then performed in real-time. Also, in addition to performing the filtering operation, statistics relating to network usage are computed and maintained.

Two examples of communication cards for use with the Sun are the MVME374 Ethernet™ Controller Module [31] and the MVME372 Advanced MAP Controller. The first provides a Sun interface to an 802.3 Ethernet™ network and the second an interface to an 802.4 token-bus network. Both these boards contain a separate processor and local memory that is used for the corresponding protocol software: the first the complete TOP [32] software suite and the second the complete MAP suite. With the analyser, however, the latter is not used, of course, and instead the physical network interface controller operates in the promiscuous mode and a small program is then used that simply performs the filtering and statistics gathering operations.

Both boards can monitor the cable in real-time and deduce frame exchanges between two specified systems. The frame exchanges between the two specified systems are passed to the host system for processing. The frames are time stamped and stored on disc to await the decode and analysis processing. In addition, the frame contents are printed on one of two windows: one to display all the frames sent by the source node and the other to display all the frames sent by the destination node. In this way, the operator can readily see that the system is performing the *Monitoring* operation and, hopefully, both systems are exchanging frames.

For test and development of the analyser two different approaches were used. The first used a set of programs (one per layer) that were written to allow a user to interactively create composite frames comprising any selection of layer PDUs. A set of such frames were then used for the earlier test and development phases.

In the second approach the analyser was tested with real data gathered from the communication between two Sun Workstations® running the MAP3.0 software. A subset of these frames was shown earlier in Fig. 7 and all the analysis operations were performed on this data to verify its correct operation.

7 CONCLUSIONS

The aim of the work described in this paper has been to develop a suite of software tools to aid the detailed analysis of the interactions between two computers connected to an OSI network that is based on the ISO protocols. The results of the analysis operation are presented to the user against suitable graphical templates. Also, the level of detail in the presentation of the results can be interactively controlled. The result is a high-level, user friendly interface that requires a minimum of knowledge of protocols on the part of the operator.

In addition to being used with the analyser, some of the software tools described are also being used as the basis of other instruments. For example, a Dynamic Tester [33] is currently being implemented that allows a suspected faulty system to be tested dynamically and the cause of the fault diagnosed automatically. This employs the same Environment Selection software and also the same *Monitoring* and *Message Decode* Run-Time components. Selected test scenarios each comprising composite frame sequences are first created, together with the expected response frames, using these tools. The actual and expected response frames are then compared and any differences highlighted against the corresponding PDU templates.

8 ACKNOWLEDGEMENTS

The work reported in this paper has been partially funded by The Networking Centre Ltd. of Hemel Hempstead, England. The author would like to acknowledge this support and also the many usefull discussions with them about the facility affered by the analyser.

9 REFERENCES

[1] EXCELAN: 'Excelan Networking', IEEE Spectrum, 1988.
[2] NETWORKII.5: 'NetworkII.5', IEEE Spectrum, 1987.
[3] Spider Systems: 'SPIDERMONITOR, Ethernet Monitor', Spider Systems Ltd, 1987.
[4] Hewlett Packard: 'HP 4951C and HP 4952A Protocol Analysers', Hewlett Packard Inc., 1985.
[5] Hewlett Packard: 'X.25 Network Performance Analyser for the HP 4953A', Hewlett Packard Inc., 1985.
[6] Trend Data Link: 'INTERVIEW 7000 Series Protocol Analysers', Atlantic Research Corporation, 1987.
[7] Network General: 'The Sniffer, The Case of 'Slower is Faster'', Network General Corporation, 1987.
[8] ISO 7498 (1988), OSI: Basic ReferenceModel for OSI.
[9] ISO 7498-1 PDAD 2 (1988), Part 1,Multipeer Transmission.
[10] ISO/DIS 7498-2 (1988), OSI: Final Text ofPart 2, Security Architecture.
[11] ISO/DIS 7498-3 (1987), Part 3, Naming and Addressing.
[12] ISO/DIS 7498-4 (1988), Part 4, Management Framework .
[13] ISO 8072 (1987), OSI: Transport Service Definition.
[14] ISO 8073 (1987), OSI: Transport Protocol Specification.
[15] Modiri, N.: 'Graphical Presentation of Protocol Behaviour', The University of Sussex, Internal Report, 1987.
[16] Modiri, N., Halsall F.: 'An Analyser for Monitoring and Analysis of OSI Protocol', NETWORKS '88, 1988.
[17] General Motors Corporation: 'Manufacturing Automation Protocol', Manufacturing Engineering and Development, Advanced Product and Manufacturing Engineering Staff (APMES), GM Technical Centre. 1988.
[18] OSI: File Transfer, Access and Management, Final Text:
DIS 8571-1 (1988), Part 1, General Introduction.
DIS 8571-2 (1988), Part 2, Virtual Filestore Definition.
DIS 8571-1 (1988), Part 3, File Service Definition.
DIS 8571-4 (1988), Part 4, File Protocol Specification.
[19] ISO 8326 (1987), OSI: Basic Connection Oriented Session Service Definition.
[20] ISO 8326 (1987), OSI: Basic Connection Oriented Session Protocol/Specification.
[21] ISO 8822 (1987), OSI: Connection-Oriented Presentation Service Definition.
[22] ISO 8823 (1987), Connection-Oriented Presentation Protocol Specification.
[23] ISO 8824 (1987), OSI: Specification of Abstract Syntax Notation One (ASN.1).
[24] ISO 8824 PDAD 1 (1987): ASN.1 extensions to ASN.1

[25] ISO 8825 (1987), OSI: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).
[26] ISO 8825 PDAD 1 (1987): ASN.1 extensions to ASN.1 Basic Encoding Rules
[27] England D.: 'A Sun User's Guide', Macmillan Education Ltd., 1987.
[28] Kernighan, B.W., Ritchie, D.M.: 'The C Programming Language', Prentice-Hall, 1987.
[29] ISO/DIS 9506/1: Manufacturing Message Specification - Service Definition.
[30] ISO/DIS 9506/2: Manufacturing Message Specification - Protocol Specification.
[31] Motorola: 'Motorola and VMEbus' Motorola Computersysteme, 1987.
[32] Farawich, S.A.: 'Communicating in the Technical Office', IEEE Spectrum, 1986.
[33] Halsall, F., Modiri, N.: 'A Dynamic Tester For Use With OSI Networks', Submitted to Telecommunication for Publications, May, 1989.

UNIX™ is a registered trademark of AT & T and Bell Laboratories.
Sun Workstation® is a registered trademark of Sun Microsystems Inc.
Ethernet™ is a registered trademark of the Xerox Corporation.

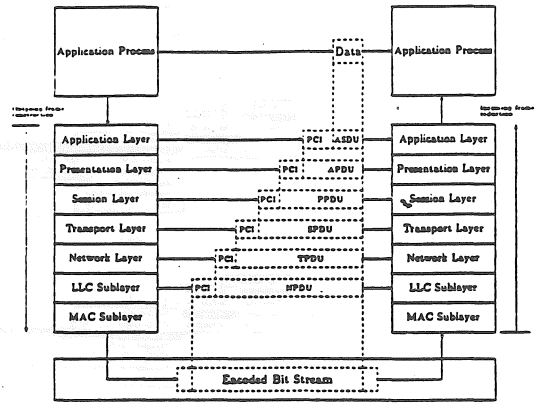


Figure 1: *Multiple Layer Interactions.*

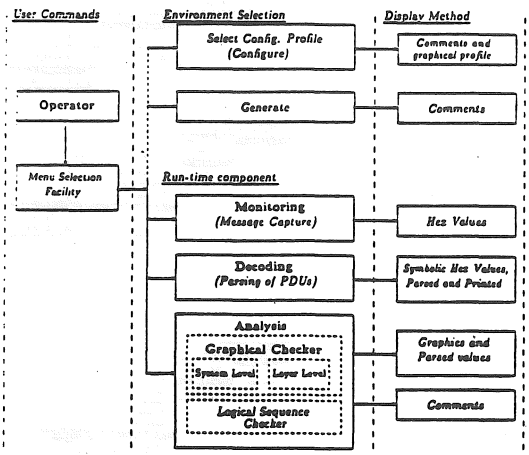


Figure 2: *Functional Representation of the Network Protocol Analyser.*

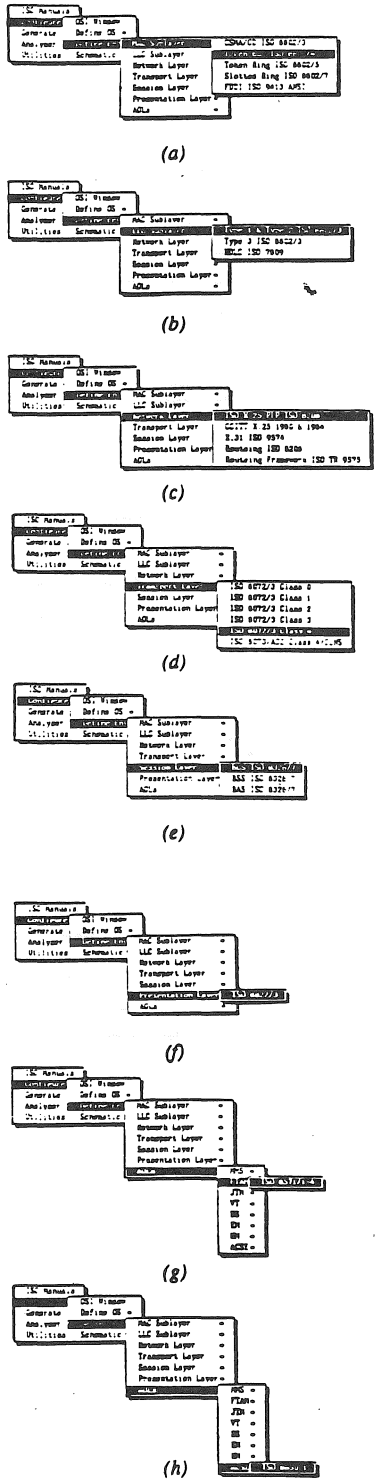
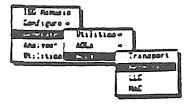
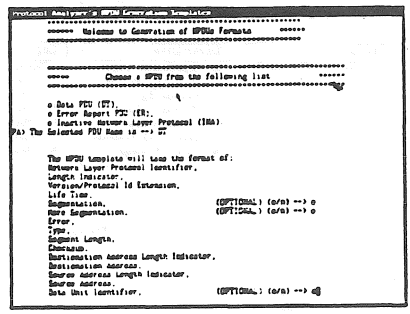


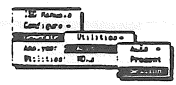
Figure 3: Configuration Profile: (a) MAC Sublayer; (b) LLC Sublayer; (c) Network Layer; (d) Transport Layer; (e) Session Layer; (f) Presentation Layer; (g) ACSE Application Entity; (h) FTAM Application Entity.



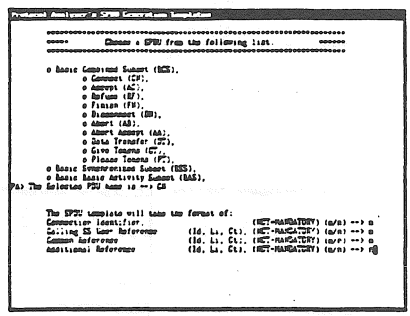
(a)



(b)



(c)



(d)

Figure 4: PDU Template Generation: (a) Select Network Layer; (b) Define DT NPDU; (c) Select Session Layer;(d) Define CN SPDU.

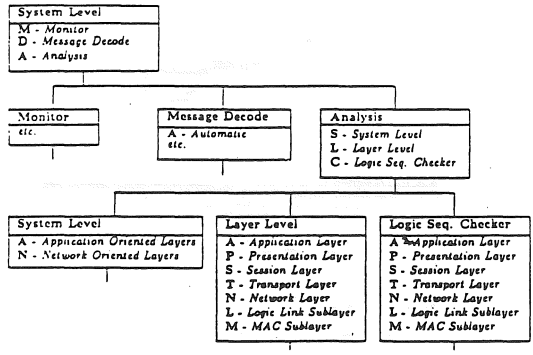


Figure 5: *Hierarchical Menu of the Network Protocol Analyser.*

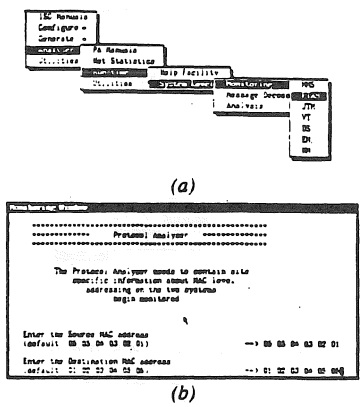


Figure 6: Monitoring Phase: (a) Select FTAM Application Entity; (b) Specify Source and Destination Address.

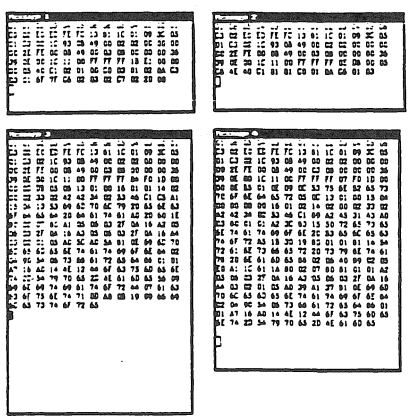
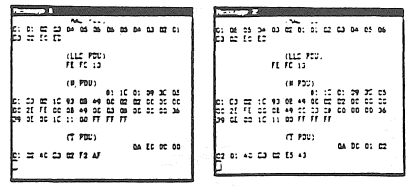
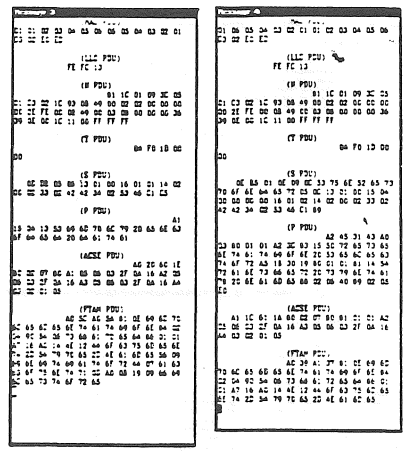


Figure 7: *Typical Message Sequence.*



(a)



(b)

Figure 8: *PDU Boundaries: (a) Transport Layer Frames; (b) FTAM Application Entity.*

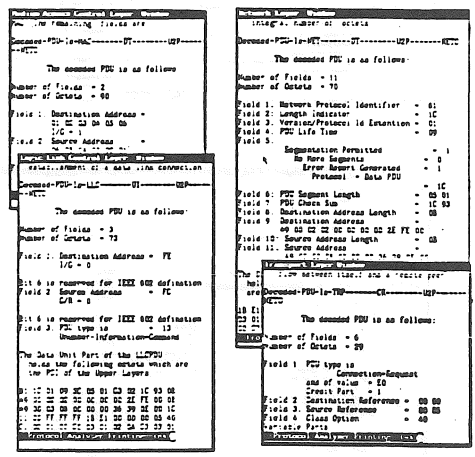
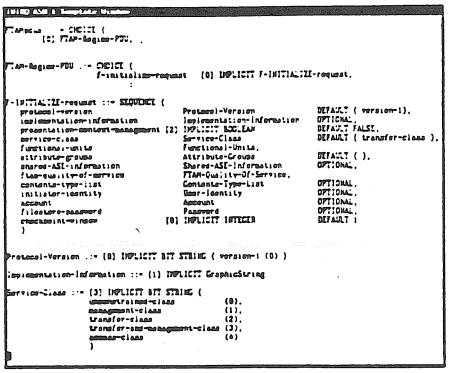
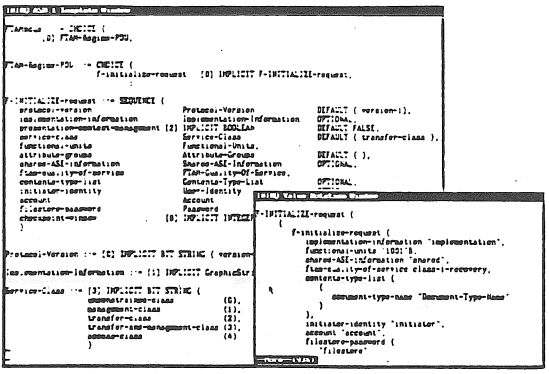


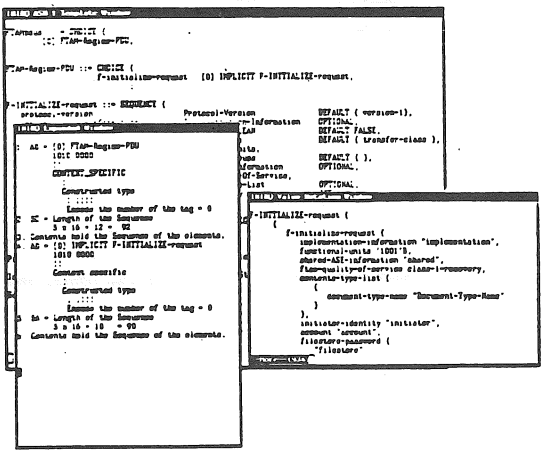
Figure 9: ***First Message Block Parsed.***



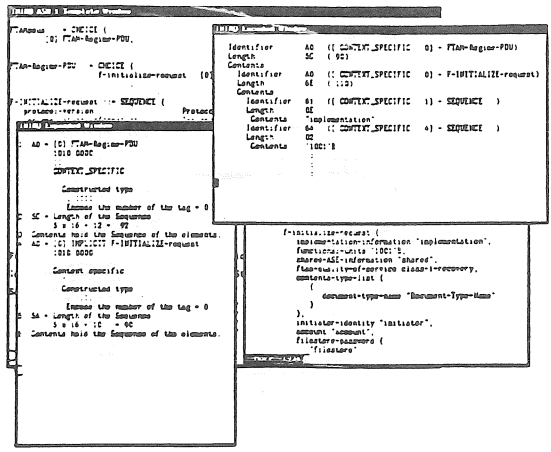
(a)



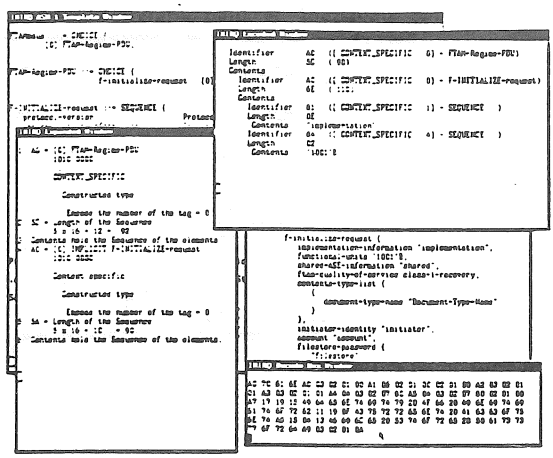
(b)



(c)



(d)



(e)

Figure 10: Encoding FTAM INIRQ PDU: (a) INIRQ PDU; (b) Concrete Value Notation; (c) Encoding Procedure; (d) INIRQ contents; (e) Concrete Syntax:

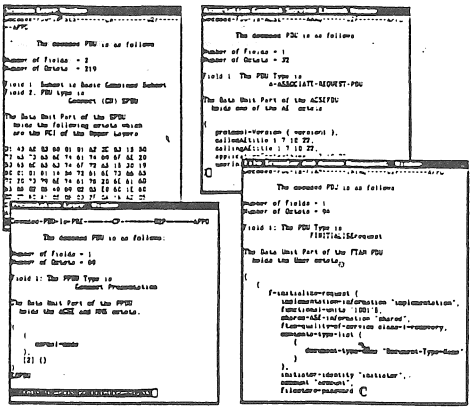
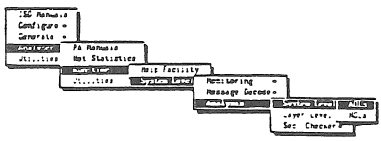
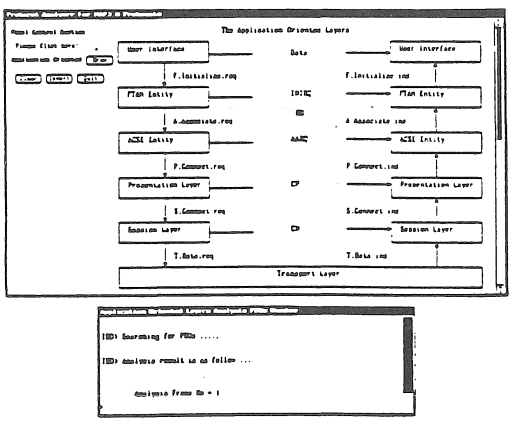


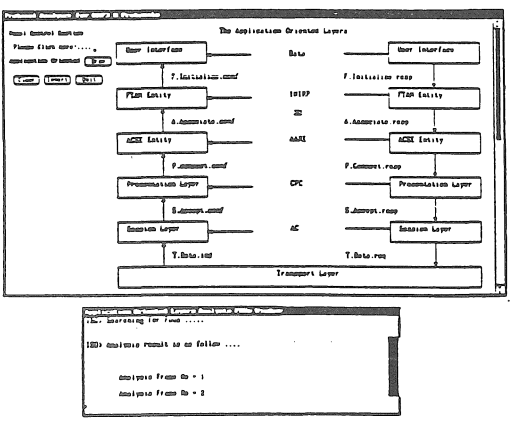
Figure 11: *Second Message Block Parsed.*



(a)

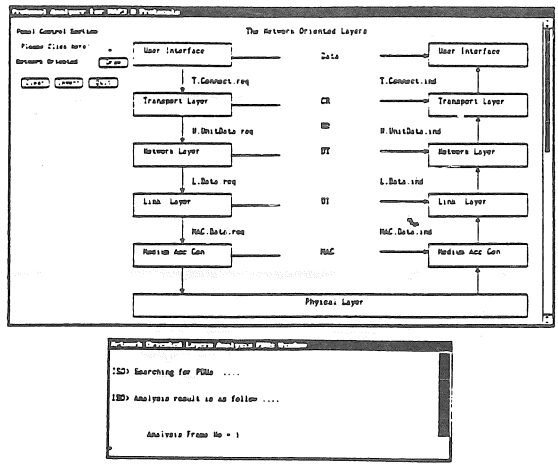


(b)

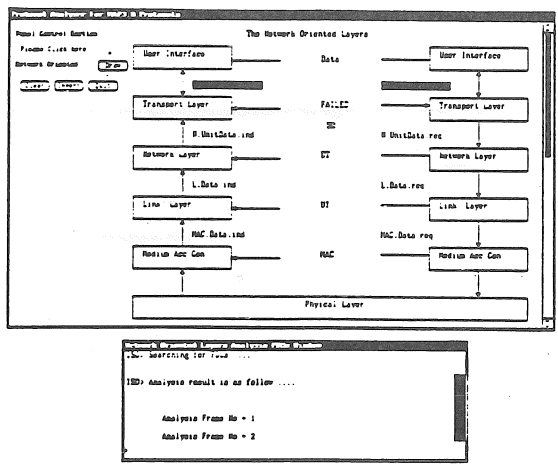


(c)

Figure 12: Analysis of the Aplication Oriented Layers: (a) Select FTAM Application Entity; (b) Initialising Frame; (c) Responding Frame.

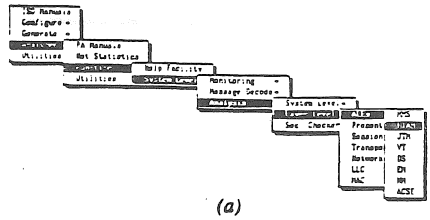


(a)

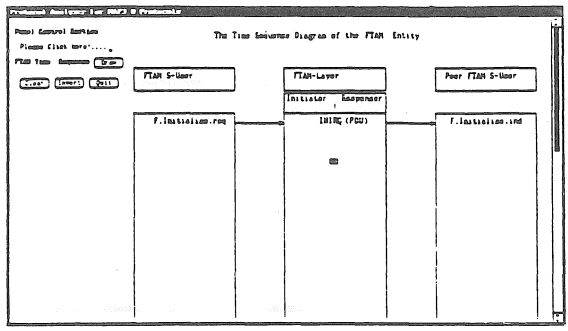


(b)

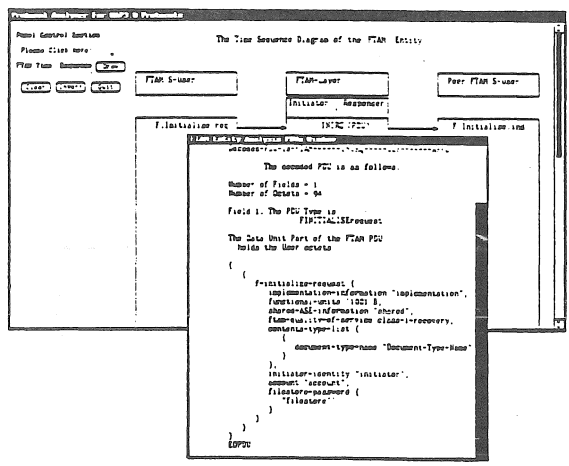
Figure 13: Analysis of the Network Oriented Layers: (a) Connection Request Frame; (b) Connection Confirm Frame.



(a)



(b)



(c)

Figure 14: Layer Level:(a) Select FTAM Application Entity; (b) Time Sequencing Diagram; (c) The Parsed INIRQ FTAM PDU.

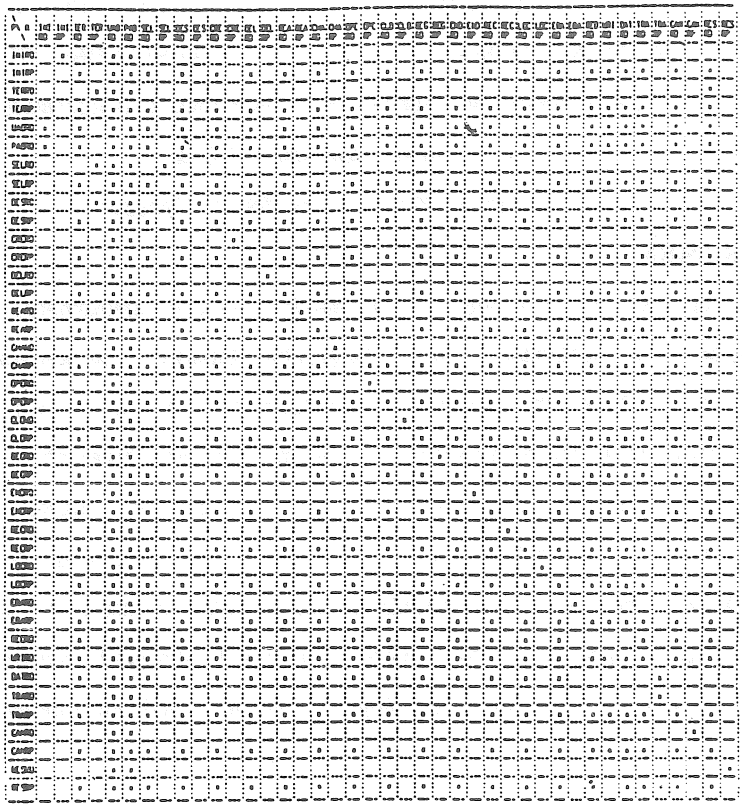


Figure 15. *Sequence of the FTAM PDU.*

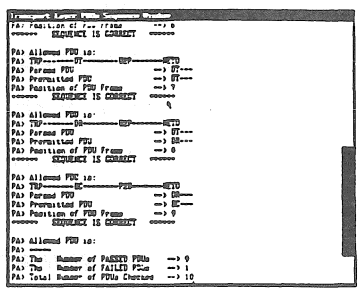


Figure 16: *LSC of the Transport Layer PDUs.*

ALGORITMOS DE COMUNICACION EN HIPERCUBOS DE BASE B

CARMEN B. MORA ALEJANDRO TERUEL

Departamento de Computación y Tecnología de la Información

Universidad Simón Bolívar

Apartado 8900, Caracas 1080-A, Venezuela

RESUMEN

Una tópologia de interconexión de multiproceadores ampliamente usada por su relación costo-efectividad es la de hipercubos binarios. Una generalización de la estructura de hipercubos binarios es la de hipercubos de base b. En este trabajo se presenta el desarrollo de algoritmos de comunicación para hipercubos de base b. Los algoritmos obtenidos presentan resultados favorables en cuanto a tiempos de comunicación a medida en que aumenta la base. La tecnología disponible en la actualidad limita en cierta medida la implementación física de las estructuras generalizadas dados los costos mayores en nodos de alto grado, sin embargo, no hay que descartar que los avances tecnológicos definan la factibilidad de implementación a futuro permitiendo aprovechar las ventajas de comunicación.

1-INTRODUCCION

La comunicación entre procesos es frecuentemente uno de los principales obstáculos para incrementar el rendimiento de los algoritmos paralelos en multiprocesadores. Saad y Shultz presentan en [5] y [6] algunos algoritmos para intercambio de datos entre procesadores en distintas topologías, destacándose los hipercubos binarios. Las redes de procesadores basadas en hipercubos son atractivas debido a sus propiedades de interconexión entre procesadores.

Lakshmivarahan y Dhall, presentan en [3] una generalización de la topologia de hipercubos, llamada hipercubos de base b, los cuales sugieren son más tolerantes a fallas en la medida en que aumenta la base. Esto se fundamenta en el hecho de que hipercubos con mayor base presentan mayor cantidad de caminos paralelos. Otra propiedad importante, es la reducción del diámetro (distancia mínima entre dos nodos

más lejanos) en estas estructuras a medida en que aumenta la base, lo cual sugiere la posibilidad de incrementar la velocidad en la comunicación entre nodos con respecto a los hipercubos binarios. Esto es debido a que los tiempos de comunicación en muchos de los algoritmos sobre hipercubos son proporcionales al diámetro de la estructura.

Los objetivos de este trabajo han sido desarrollar para base genérica b, algunos algoritmos de comunicación existentes para hipercubos binarios, así como algoritmos que surgen como necesidades en la práctica, y analizar el rendimiento de hipercubos de mayor base.

2- LAS ESTRUCTURAS DE HIPERCUBOS

Las definiciones y propiedades que se presentan en esta sección se basan en los trabajos [7] y [3] donde se introducen las estructuras de hipercubos binarios e hipercubos en base b respectivamente. Dadas las restricciones de espacio, se supondrán conocidas las propiedades de los hipercubos binarios.

Extenderemos la definición de distancia de Hamming para considerar las etiquetas en base b, donde b > 1. La representación en base b de cualquier entero $0 <= X < k$ puede ser escrita como $X_1X_2..X_N$, X_i <b>, donde <b> = {0,1,2,..,b-1}. La distancia de Hamming en base b, denotada $H_b(A,B)$, entre dos nodos A,B es el número de dígitos en base b en que difieren A y B.

Hipercubo de base b. Un hipercubo de base b, y dimensión N, consiste de $k=b^N$ nodos, $2 <= b <= k$, donde dos nodos están conectados si y sólo si sus etiquetas en base b difieren en uno y sólo un dígito. Esta definición corresponde a la de hipercubo completo de dimensión N y base b, presentada en [3]. La Figura 1 presenta un ejemplo de hipercubo de 9 nodos, base 3 y dimensión 2.

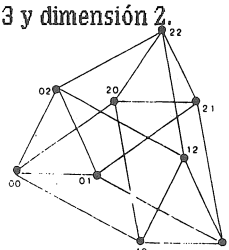


Figura 1. Ejemplo de hipercubo de base 3, k=9, N=2

3- ALGORITMOS DE COMUNICACION

Los algoritmos que fueron desarrollados para base genérica b, están basados en las presentaciones hechas para el caso de hipercubos binarios en [1], [2] y [5]. Los algoritmos seleccionados para su generalización a base b son: Envío de datos entre dos nodos, Divulgación (Broadcast), Dispersión (Scattering), Recolección (Gathering), Intercambio total, Multienvíos (Multicasting) y Transposición de datos. A continuación se presenta una breve descripción de cada problema de comunicación y seguidadmente una selección de algoritmos que representan mejoras sustanciales en tiempos de comunicación con respecto al caso binario. Los algoritmos restantes se encuentran desarrollados en [4].

ENVIO DE DATOS ENTRE DOS NODOS

El envío de datos desde un procesador a otro representa la operación de transferencia de datos más simple entre los algoritmos de comunicación propuestos. La transferencia de datos se lleva a cabo a través del pase de mensajes entre nodos vecinos.

DIVULGACION

El problema de divulgación se refiere a la transferencia de datos desde un nodo en particular a todos los otros nodos del hipercubo.

RECOLECCION

La recolección o "Gathering", se refiere a la transferencia de la misma cantidad de datos desde todos los nodos a un nodo en particular. En este caso cada nodo envía un paquete de datos distinto al nodo receptor.

DISPERSION

El problema de dispersión o "scattering", consiste en la transferencia de datos distintos desde un nodo en particular a todos los otros nodos del hipercubo. A diferencia del problema de divulgación, los datos que recibe cada nodo son distintos, por lo que hay que controlar en el algoritmo, la distribución correcta de los datos a los nodos correspondientes.

INTERCAMBIO TOTAL

El intercambio total de datos se refiere al envio de paquetes iguales desde cada nodo hacia todos los otros nodos. Esta operación puede verse como multidivulgación puesto que cada nodo necesita hacer divulgación de su dato a todos los otros nodos.

MULTIENVIOS

La forma de comunicación de datos por multienvios, se refiere al tipo de comunicación entre procesos en el cual un nodo fuente desea enviar un mismo mensaje a m distintos nodos destinos. Esto constituye una generalización con respecto a la comunicación referida al envío de datos entre dos nodos (m=1) y a la de divulgación ($m=2^N-1$).

TRANSPOSICION DE DATOS

El problema de transposición de datos se refiere al envío de datos distintos desde cada procesador hacia todos los otros procesadores. Esto puede ser considerado como una generalización del problema de dispersión o en forma análoga, del problema de recolección.

RESULTADOS

A efectos del diseño y análisis de rendimiento de los distintos algoritmos de comunicación desarrollados, se consideró el siguiente modelo:

- Todos los paquetes tienen igual longitud
- El tiempo para transmitir un paquete desde un nodo cualquiera a un nodo vecino es $\beta + \mu$ donde:

  β = Latencia o tiempo de inicio de comunicación

  μ = Tiempo que tarda un paquete en atravezar un canal de comunicación.
- El espacio de almacenamiento de cada nodo es infinito. Esto implica que no se consideran retardos ocasionados por insuficiencia de espacio de almacenamiento en nodos receptores.
- La transmisión de un paquete desde un nodo a cualquier número de nodos vecinos se efectúa en la misma cantidad de tiempo.
- Todos los canales de comunicación incidentes a un nodo pueden ser usados simultáneamente para transmisión y recepción de paquetes (MLA).

- Dado un número k de nodos, las estructuras de hipercubos a comparar son de la forma:

  1) Base 2, dimensión $n = \log_2 k$, $(k=2^n)$

  2) Base b, dimensión $N = \log_b k$, $(k=b^N)$

Los distintos algoritmos pueden se agrupados según cuatro técnicas distintas que representan la forma en que fué atacado cada problema. En la Tabla 1 se presentan los resultados obtenidos en cuanto a tiempos de ejecución, en forma comparativa para base 2 y base b.

ALGORITMO	TECNICA	BINARIO	BASE b
Envío entre dos nodos Divulgación Recolección Dispersión	Envío Simultáneo	$n(\beta+\mu)$ $n(\beta+\mu)$ $n\beta+(k/2)\mu$	$N(\beta+\mu)$ $N(\beta+\mu)$ $N\beta+(k/b)\mu$
Divulgación Recolección Dispersión Transposición	Recursión	$n(\beta+\mu)$ $n\beta+(k-1)\mu$ $n\beta+(k-1)\mu$ $(k-1)$	$N(\beta+\mu)$ $N\beta+(k-1/b-1)\mu$ $N\beta+(k-1/b-1)\mu$ $(k-1)!/(b-1)^{-1}$
Intercambio Total Transposición	Partición	$n\beta+(k-1)\mu$ $n(\beta+(k/2)\mu)$	$N\beta+(k-1/b-1)\mu$ $N(\beta+(k/b)\mu)$
Multienvíos	Dimensión Común	$n(\beta+\mu)$	$N(\beta+\mu)$

Tabla 1. Algoritmos desarrollados/resultados

La técnica de envío simultáneo para la construcción de algoritmos de comunicación se refiere a la idea de transmitir cada paquete de datos haciendo uso de la mayor cantidad de canales de comunicación disponibles y enviar en forma simultánea por cada uno de ellos.

La técnica de recursión se refiere a la construcción del algoritmo siguiendo los caminos de comunicación que resultan de la construcción recursiva de hipercubos a partir de hipercubos de dimensión menor.

La construcción de algoritmos de comunicación según la técnica de partición, se refiere a utilizar en cada paso los canales de comunicación que definen la partición del hipercubo según alguna dimensión.

La técnica de dimensión común busca enrutar los paquetes de datos por los enlaces de comunicación correspondientes a la dimensión del hipercubo donde hay mayor cantidad de nodos receptores.

A título de ejemplo, se presenta a continuación uno de los algoritmos desarrollados según la técnica de partición.

INTERCAMBIO TOTAL (Técnica: Partición)

El algoritmo de intercambio según dimensiones presentado en [5] para el caso binario como ADEA (Alternate Direction Exchange Algorithm), consiste en dividir en cada paso, según una dirección distinta, el cubo de dimensión n en dos cubos de dimensión n-1 y realizar el intercambio de datos entre los nodos opuestos correspondientes de los dos cubos de dimensión menor. La correspondencia entre nodos referida como "nodos opuestos" según la dirección i-ésima se define de la siguiente manera: Dos nodos vecinos A y B son opuestos según la dirección i, si las etiquetas A y B difieren en la posición i-ésima. Sin pérdida de generalidad, se toma como sentido de dirección desde el bit menos significativo al más significativo (derecha a izquierda). La Figura 2 presenta un esquema de como se realiza el intercambio para el caso de un hipercubo binario de dimensión 3.

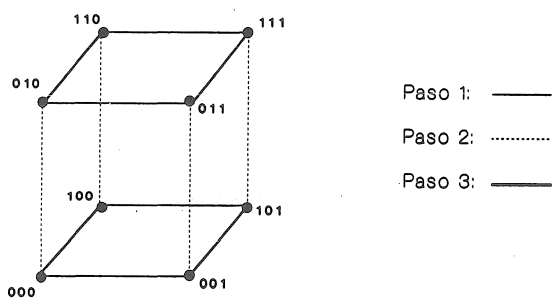


Figura 2. Intercambio según ADEA en hipercubo de dimensión 3.

Para el algoritmo correspondiente a base b, debe considerarse que ahora cada nodo tiene b-1 vecinos opuestos según cada dirección. Esto se debe a que en base b, cada dígito puede variar según b posibles valores.

ADEA en hipercubos base b

Sea $N = \log_b k$

For i=1 to N do

Cada nodo intercambia sus b^{i-1} datos acumulados con los b-1 nodos opuestos en la i-ésima dirección

Cabe observar que el intercambio del lazo interno no es un intercambio entre dos nodos sino entre b nodos. Sin embargo, dado que estos b nodos son vecinos entre sí (todos difieren en la misma posición i), y haciendo uso de la suposición MLA, este intercambio entre b nodos puede realizarse en 1 sólo paso. Bajo esta consideración el ciclo interno tendrá entonces una duración asociada a la cantidad de datos a intercambiar y no a la cantidad de nodos que intervienen en el intercambio.

CONCLUSIONES

El aporte fundamental del presente trabajo ha sido el desarrollo de las rutinas básicas de comunicación para hipercubos de base genérica b, basado en los algoritmos existentes para hipercubos binarios. Dichos algoritmos han sido diseñados para ser usados sin cambios con cualquier valor de base b >= 2, aprovechando la disponibilidad de canales de comunicación adicionales según aumenta el valor de la base y logrando en consecuencia mejoras en los tiempos de ejecución.

Como resultados del desarrollo de algoritmos para base genérica b, se pudo comprobar la hipótesis de incidencia favorable de dimensiones menores en los tiempos de comunicación entre los procesadores.

Los algoritmos desarrollados constituyen un punto de partida para la construcción de una librería de subrutinas que realicen las tareas de comunicación más comunes en sistemas de multiprocesadores. Aunque hasta el momento no se conoce la existencia de estas arquitecturas, queda pues para futuros trabajos la implementación y prueba de

dichas rutinas en un sistema simulado. Todo parece indicar que la tecnología disponible limite la conveniencia de altos grados en los nodos, y en consecuencia las ventajas de una mayor base. La tendencia es una vuelta hacia estructuras más sencillas. La disponibilidad en la actualidad de conmutadores de reconfiguración que permiten modificar la topología de interconexión en forma dinámica, permitiría decidir según la aplicación, el tipo de configuración que más se adecúe a la naturaleza del problema. No hay que descartar sin embargo que los avances tecnológicos en los próximos años permitan disponer a bajo costo de nodos de alto grado de comunicación, y es de esperarse la oportunidad para las estructuras de hipercubos generalizadas.

REFERENCIAS

[1] D. P. Bertsekas, C. Ozveren, G.D. Stamoulis, P. Tseng y J. N. Tsitsiklis, "Optimal Communication Algorithms for Hypercubes", Journal of Parallel and Distribuited Computing, 11, p. 263-275, (1991).

[2] A. H. Esfahanian y M. L. Ni, "Multicast in Hypercube Multiprocessors", Journal of Parallel and Distributed Computing, 8, p.30-41, (May 1988).

[3] S. Lakshmivarahan y S. K. Dhall, "A New Hierarchy of Hypercube Interconnection Schemes for Parallel Computers", The Journal of Supercomputing, 2, p. 81-108, (1988).

[4] C. Mora , "Algoritmos de Comunicación en Hipercubos de Base b", Tesis de Maestría en Ciencias de la Computación", Universidad Simón Bolívar, (Enero 1992).

[5] Y. Saad y M. H. Schulz, "Data Communication in Hypercubes", Journal of Parallel and Distribuited Computing, 6, p. 115-135, (1989).

[6] Y. Saad y M. H. Schulz, "Data Communication in Parallel Architectures", Research Report YALEU/DSC/RR-461, Yale University, (March 1986).

[7] Y. Saad y M. H. Schulz, "Topological Properties of Hypercubes", Research Report YALEU/DSC/RR-389, Yale University, (June 1985).

Un Marco Formal para la Programación Implícita de Robots

Jorge Elías Morales V.
Jorge A. Villalobos S.

Grupo DFAC
Diseño y Fabricación Asistidos por Computador
Departamento de Ingeniería de Sistemas y Computación
Universidad de los Andes
Apartado Aéreo 4976 -Bogotá, Colombia
e-mail: jvillalo@andescol.bitnet

Resumen: Se presenta una formalización de los conceptos usados para describir los elementos del mundo de operación de un robot. Dichos conceptos pueden ser usados en una presentación formal de un lenguaje implícito de programación, basado en especificación de planes de solución de tareas, en términos de acciones, las cuales son traducidas en operaciones de robot [8]. Además, se incorpora el concepto de rasgos de ensamblaje (features), como información adicional para facilitar el proceso de traducción de las tareas de ensamble, tal y como está en las propuestas de manejo de rasgos de [7], [2] y [10].

Palabras clave: robótica, programación de robots, lenguajes implícitos, rasgos, planes de solución.

1. Introducción

El problema de la programación de robots se ha convertido en elemento clave del proceso de automatización industrial, porque de la facilidad de dicha programación, depende el éxito de incorporarlos en el desarrollo industrial del país. Por esta razón, se ha venido investigando en esta área: desde el desarrollo de ambientes de programación de robots [11], hasta herramientas autómaticas de programación [8].

En este artículo se presenta un marco formal para el desarrollo de este tipo de herramientas, siguiendo el esquema de programación asistida por el programador. Utiliza un esquema de descripción de planes de solución e información de interés para el proceso de programación (rasgos de ensamble). A partir de esta especificación, es posible diseñar un modelo de compilador para un lenguaje automático y presentar un paradigma de programación de robots [8]. La estructura de este artículo es la siguiente: inicialmente se presenta el concepto de rasgo de ensamble, luego se presentan los conceptos a través de los cuales se construye el modelo del mundo y finalmente, se define validez de los elementos.

2. Rasgos

Los rasgos surgen como una propuesta para enriquecer las herramientas que apoyan el proceso de diseño (CAD/CAM), de tal forma que sea más natural para el diseñador expresarse. Consiste en agregar información que caracterice los elementos del proceso, facilitando la expresión de las acciones propias de éste. Una primera propuesta formal se presenta en [7] [2] y una especificación formal de un sistema basado en rasgos en [10].

Los rasgos (features) son características de los objetos, adicionales a la geometría, para manejar información que facilite el proceso en el que está involucrado este objeto. Dicha información caracteriza clases de objetos por comportamiento, forma, etc., adquiriendo una identidad, de tal forma que permiten expresar la semántica del proceso a través de ellos y facilitan la labor de los usarios finales. Dentro del proceso de ensamblaje de piezas metalmecánicas (rígidas), utilizando robots, los rasgos expresan información que facilita las operaciones necesarias del robot para resolver una tarea de ensamblaje, como agarrar un objeto por un punto estable, transportarlo, ensamblarlo con otros objetos, etc.

2.1. Clasificación de rasgos de ensamblaje

Rasgo de forma: Es un conjunto de superficies pertenecientes a un objeto o ensamble que definen una característica interesante para el proceso de ensamblaje. Estos rasgos se caracterizan con un identificador, un sistema de referencia que define la localización del rasgo sobre el objeto y un conjunto de identificadores de objetos con los que se puede relacionar el objeto a través de este rasgo.

Rasgo de manipulación: Característica importante de un objeto, útil para el manejo de éste utilizando un robot. Estos rasgos se caracterizan con un identificador y un sistema de referencia que define la localización del rasgo sobre el objeto. Un ejemplo es un punto de agarre estable de una pieza.

Rasgo físico: Característica física de un objeto o ensamble. Se usa para informar a los procesos de planificación de ensamblaje condiciones de estabilidad, centro de masa, etc. Estos rasgos se caracterizan con un identificador, un indicador del tipo de característica física y un valor asociado con esa característica.

Rasgo funcional: Describe la semántica de un objeto o ensamble. Se usa para verificar la validez de las acciones realizadas por el proceso de ensamblaje. Se caracterizan con un identificador, un verbo que expresa la acción que realiza el objeto y un conjunto de objetos con los que está asociado el objeto a través de dicha funcionalidad.

2.2. Especificación de rasgos

A continuación se especifica un conjunto de rasgos base, usados para la especificación del modelo del mundo y su validez. La caracterización de este conjunto de rasgos, en particular, está dada para robots manipuladores de hasta seis (6) grados de libertad, usando una pinza de dos dedos paralelos, sin tener en cuenta características físicas como peso, estabilidad, inercia, etc.

Sección de ensamblaje: Es un rasgo de forma que consiste en un conjunto de superficies, que entran en contacto con una sección de ensamblaje de otro objeto o ensamble, estableciendo una relación de unión entre ellos. A partir de esta relación de unión, los objetos se hacen un solo ensamble.

Superficie de soporte: Una superficie de soporte es un rasgo de manipulación que consiste en un conjunto de superficies especificadas a través de un sistema de referencia que sirve para apoyar el objeto sobre otra superficie, manteniéndose estable.

Punto de agarre: Un punto de agarre es un rasgo de manipulación que indica una posición estable para agarrar un objeto o ensamble usando un robot. La caracterización de un punto de agarre es un identificador, un sistema de referencia que define la localización del punto sobre el objeto y una apertura de la pinza.

3. Conceptos básicos

A continuación, se definen los conceptos básicos para la especificación de los elementos del mundo de operación de un robot. Sea INDEF un constante que evalúa indefinido para la especificación.

Un robot: Un robot industrial R es una secuencia de N cuerpos o eslabones rígidos (grados de libertad) en forma de cadena cinemática abierta simple, a través de N pares rotacionales o prismáticos. Está caracterizado por una descripción geométrica de cada eslabón y por una descripción cinemática. Se representa por una tupla de la forma: R = < nombre, N, G, C, DH, [min1, max1], ..., [minN, maxN], P >, donde N es el número de grados de libertad, G es la geometría, C es el modelo cinemático, DH son los parámetros de Denavit-Hartenberg, [min1, max1], ..., [minN, maxN] son los valores articulares máximos y mínimos para cada articulación y P es la pinza o herramienta que usa el robot.

Variable de estado de un robot R es el conjunto de valores que definen un estado único del robot en un momento dado, en términos de su posición y de si lleva un objeto en la pinza. Se representa por una tupla de la forma VER = < R, Ø1, ..., ØN, Apertura, Obj >, donde R es el robot, Ø1,...,ØN son los valores articulares del robot, Apertura define la apertura de la pinza y Obj es el objeto (si lleva) en la pinza.

Un obstáculo Obt del mundo es un elemento geométrico definido con respecto a un sistema de referencia local, más un conjunto de rasgos superficie de soporte. Los obstáculos se usan para describir el mundo de trabajo donde está el robot y son objetos que no pueden ser manipulados por el robot, únicamente pueden ser usados como elementos de apoyo para resolver las tareas. Un obstáculo se representa por una tupla de la forma Obt = < nombre, G, {R1, ..., Rk } >. El estado de un obstáculo es la transformación que define la ubicación del obstáculo respecto a un sistema de referencia de base. Se representa a través de una variable de estado de la forma VEObt = < Obt, T >.

Un objeto O del mundo está definido por su geometría (con respecto a un sistema de referencia local) y un conjunto de rasgos. Se representa por una tupla de la forma O = < nombre, G, { R1, ..., Rk } >. El estado de un objeto es la transformación que define la ubicación del objeto respecto a un sistema de referencia base y el ensamble al que pertenece (si hay). Se representa a través de una variable de estado de la forma VEO = < O, T, listaEnsamble >. Cuando la localización de un objeto no está definida, se asocia en su variable de estado la transformación indefinida. VEO = < O, INDEF, listaEnsamble >. Igualmente, cuando no esta definido el ensamble al que pertenece un objeto, se asocia al objeto el ensamble indefinido. VEO = < O, T, INDEF >.

Un ensamble E del mundo es una relación definida para un conjunto de objetos, asociados explícitamente bajo un nombre NE, en el cual un conjunto de superficies de cada objeto en el ensamble entra en contacto con superficies de otro objeto o ensamble en NE, de tal forma que se comporten en forma sólidaria. Cuenta con un sistema de referencia local, una transformación que lo ubica en el sistema de referencia del mundo y por un conjunto de rasgos del ensamble. Está definido por el conjunto de objetos que lo constituye más un conjunto de rasgos. Se representa por una tupla de la forma E = < NE, O1, ..., On, { R1, ..., Rk } >. El estado de un ensamble es la transformación que define la ubicación del ensamble respecto a un sistema de referencia base (el mundo), durante una tarea dada. Se

representa a través de una variable de estado definido por una tupla de la forma VEE = < E, T >. Donde T es una transformación que define la localización de cada uno de los objetos en el ensamble de la forma Localización (Oi) = T.TOi Para todo Oi en E. Cuando la localización del ensamble no está definida, no no lo está para ninguno de los objetos en el ensamble. VEE = < E, INDEF> —> Localización (Oi) = INDEF Para todo Oi en E

Un escenario ESC es un sistema de referencia SR y un conjunto de obstáculos que describen los elementos estáticos en el ambiente de trabajo de un robot. Se representa como una tupla de la forma ESC = < SR, { Obt1, ..., Obtn } >. El estado de un escenario se describe a través de un conjunto de variables de estado de obstáculos, que definen la localización de los obstáculos con respecto al sistema de referencia SR. La variable de estado de un escenario es una tupla de la forma VEESC = < ESC, { VEObt1, ..., VEObtn } >.

El mundo M es un conjunto de objetos y un conjunto de ensambles dado un escenario, que describe el ambiente donde se resolverá una tarea con los objetos descritos. El mundo es usado para crear un modelo de las tareas de ensamblaje que se vayan a resolver y se representa como una tupla de la forma M = < ESC, { O1, ..., On }, { E1, ..., Em } >.

La variable de estado del mundo VEM es un conjunto de variables de estado de los objetos y de los ensambles, dada una variable de estado de un escenario. El escenario tiene una variable de estado VEESC con un valor constante para todos los valores de las variables de estado de los objetos y ensambles en el mundo.

Una escena E es el estado del modelo del mundo M en un instante de tiempo dado. Equivale a una valuación de la variable de estado del mundo con respecto al tiempo t, dado en alguna unidad: E = VEM / t

Un bosquejo de escena BE es una escena en la cual se encuentran indefinidas algunas características de los objetos en el mundo, tales como su localización o la definición de ensambles.

Una tarea atómica de ensamblaje TA es una transformación del mundo M donde solo se modifica la localización de un objeto y ninguna o varias relaciones entre ellos. Está definida por dos escenas del mundo: TA = { E1, E2 }. TA puede ser cualquier transformación del mundo obtenida por una de las siguientes acciones de ensamblaje: agarrar, soltar, insertar, mover, localizar, girar, unir o desunir.

Una tarea de ensamblaje T es una transformación del mundo M en la cual solo se modifica la localización de los objetos y las relaciones entre ellos, no su geometría ni sus propiedades. Está definida por dos escenas del mundo que describen una situación inicial y una situación final. Estas escenas inicial y final se conocen como Precondición y Postcondición de la tarea: T = { Precondición (PRE), Postcondición (POST) }

Un plan de solución P para una tarea T especificada por { PRE, POST } es una secuencia de tareas atómicas TA1 ... TAn tales que la situación inicial de TA1 es PRE, la situación final de TAn es POST y la situación final de TAi es igual a la situación inicial de TAi+1. Si PRE es un estado válido en el mundo real y POST es un estado válido en el mundo real, puede existir un plan P, tal que se cumple { PRE } P { POST }, aunque, P no necesariamente es ejecutable para todo robot R.

El programa de ensamblaje PE correspondiente a un plan P, está definido por la secuencia de acciones de ensamblaje A1 ... An asociadas con cada una de las tareas atómicas en el plan P.

Un borrador de solución B para una tarea T especificada por { PRE, POST } es una secuencia de tareas T1 ... Tn especificadas de tal forma que no definen completamente la secuencia de tareas atómicas por realizar o es una secuencia de tareas atómicas definidas mediante bosquejos de escena o a través de descripciones incompletas de acciones de ensamblaje o es una combinación de tareas atómicas con bosquejos de tareas y tareas complejas.

Un programa del robot PR es una secuencia de operaciones de movimiento sobre un robot R expresada en un lenguaje explícito, que efectúa la tarea T definida por { PRE, POST } siguiendo un plan especificado en un programa de ensamblaje P.

Programación implícita: Dado un modelo del mundo M, un robot R y una tarea T que describe el ensamblaje que se desea desarrollar, se debe construir un plan P de solución y un programa de robot PR que haga las transformaciones definidas en P para resolver T. El problema de la validez de la tarea respecto al modelo del mundo M y al mundo real se trata en la siguiente sección.

Una heurística H es una función de probabilidad f, tal que f(x) = P, donde P es el valor de probabilidad asociado al evento x. La aplicación que se le da en este artículo es la de calificar las posibles soluciones, según un criterio dado por la función, para resolver problemas de planificación de trayectorias, de forma que, indique cual es la opción con mayor probabilidad.

Un planificador Pl es una función que dada la tarea por ejecutar f, una heurística H para calificar las soluciones de f y un entero i, cumple que Pl (f, H, i) = S, donde S es la i-ésima solución de f según H.

4. Corrección y verificación de planes

El problema de corrección y verificación de planes, se trata en forma distinta a la aplicable en programación clásica de computadores, por las razones expuestas en [8]. A continuación, se presenta el esquema que se debe seguir para construir escenarios y escenas correctas y válidas respecto a una escena del mundo real, de manera que, especificando

de esta forma las tareas {PRE, POST} y generando las escenas del plan siguiendo este esquema, se garantiza al menos consistencia en el modelo.

Posición válida de un obstáculo: Un obstáculo Obt está en una posición válida P respecto al sistema de referencia del escenario ESC si la posición se define con una de las siguientes reglas:

- P está definida por T ≠ INDEF yObt no se relaciona con ningún otro obstáculo en ESC.
- P está definida por T = Obt.SSi • Obtj.SSk y Obtj está en posición válida.

Definición de un escenario: Un escenario ESC válido se define como una valuación de VEESC donde todas las posiciones de los obstáculos son válidas y se definen con respecto al sistema de referencia del escenario o con respecto a un obstáculo válido en el sistema de referencia.

Posición válida de un objeto: Una posición válida P de un objeto O es una posición del objeto definida para un escenario válido ESC, aplicando alguna de las siguientes reglas:

- P está definida por T = O.SSi • Obtj.SSk y Obtj pertenece a ESC.
- P está definida por T = O.SSi • Oj.SSk y Oj está en posición válida en ESC.
- P está definida por T = O.SEi • Oj.SEk y O está en posición válida en ESC y O se relaciona con Oj a través de SEi y Oj se relaciona con O a través de SEk.

Y no quedar en colisión con ningún elemento en posición válida (del escenario, objeto o ensamble).

Posición en conflicto: Un objeto O u obstáculo Obt está en posición en conflicto en una escena si O u Obt está en ubicado en posición válida respecto al escenario, pero O u Obt queda en colisión con otro elemento Ej en posición válida y ubicado antes que el objeto O.

Definición de una bosquejo de escena: Un bosquejo de escena válido BE se define como un bosquejo de escena donde el escenario es un escenario válido ESC y el conjunto de objetos y ensambles en el bosquejo de escena, están ubicados en: posiciones válidas respecto al escenario, posiciones válidas respecto a objetos válidos respecto al escenario, la posición INDEF.

Definición de una escena: Una escena válida E se define como un bosquejo de escena donde ningún objeto o ensamble está ubicado en la posición INDEF.

Definición de un plan: Un plan P = TA1, ..., TAn es válido si:

- Para todo TAi, TAj, ESCi = ESCj.

- Para todo TAi = { E1, E2 }, E1 y E2 son escenas válidas respecto al escenario.
- Entre TAi y TAi+1 únicamente cambia de posición un objeto o se genera o destruye un ensamble.

Definición de un borrador de plan: Un borrador de plan es válido si cumple las dos primeras reglas de la definición de un plan.

Refinamiento de un bosquejo de escena: Un bosquejo de escena se refina cuando un objeto o ensamble definido en la posición INDEF, se redefine en una posición respecto a un obstáculo, objeto o ensamble en posición válida en el bosquejo de escena. Sin embargo, a pesar de seguir siendo un bosquejo de escena válido o de haberse convertido en escena válida, la validez del bosquejo depende de su validez en el contexto en el que se halle, en este caso un borrado de plan.

Para que el bosquejo de escena siga siendo válido en el borrador, la redefinición debe cumplir que: Sea BP = T1, ..., Tn. Sin pérdida de generalidad, podemos decir que hay un tarea Ti tal que Ti = { Ej, BE }, y que Ti+1 cumple que Ti+1 = { BE, Ek }. Sea BE = VEM / t, donde VEOr = < Or, INDEF, listaEnsamble >. Si se redefine la posición de Or, en una posición válida P, en el bosquejo diferente de la posición INDEF, debe cumplir que:

- Si de Ti+1, ..., Tn la posición de Or no cambia (es INDEF), entonces la nueva posición de Or no debe quedar en posición en conflicto con nigún objeto definido entre Ti+1 y Tn.
- Si en un Ts, s > i, Or es definido en una posición P1, debe aparecer un TA entre Ts-1 y Ts, Ts' donde Or pase de la posicion P a P1 y entre Ti+1 y Ts' se cumple la regla anterior.

5. Conclusiones

La especificación del modelo está construida con base en tres rasgos básicos, secciones de ensamble, superficies de soporte y puntos de agarre, que componen una base para las operaciones permitidas, (agarrar - llevar-dejar), pero el modelo es extendible para usar más rasgos que faciliten determinados procesos.

El trabajo de formalización en el área de programación de robots, da piso al desarrollo de herramientas automáticas que faciliten el proceso de programación y permite, en un futuro, incorporar especialistas en estrategias de solución de planes.

El modelo de programación basado en planes, ofrece una gran flexibilidad, no solo para implantarlo sino para extenderlo al manejo de sistema sensorial y replanificación.

Bibliografía

[1] Basañez, L., Operation Specialists for Automatic Programming and Monitoring of Robotic Assembly Cells, Journal of Robotics and Computer Integrated Manufacturing, 1989.

[2] De la Rosa, F., Una Propuesta de Diseño Asistido por Computador de Piezas Metalmecánicas Representados por Rasgos, Universidad de Los Andes, Tesis de Magister, Junio 1991.

[3] Laugier, C., Raisonnement Geometrique et Methodes de Decision en Robotique. Aplication a la Programmation Automatique des Robots, Tesis de doctorado, Institut National Polytechnique de Grenoble, diciembre 1987.

[4] Lieberman, L., AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly, IBM Journal of Research and Development, Vol. 21, No. 4, Julio 1977.

[5] Lozano-Perez, T., Lama: A language for Automatic Mechanical Assembly, MIT.

[6] Lozano-Perez, T., The design of a Mechanical Assembly System, MIT, 1976.

[7] Manthila, M., A Modeling System for Top-Down Design of Assembled Products, IBM Journal Res. Develop, Vol. 34, No. 5, septiembre 1990.

[8] Morales, J., Un ambiente de programación de robots con capacidad de planificación de tareas, Tesis de Magister, Universidad de Los Andes, Informe Interno, 1991.

[9] Pertin, J., Modelisation du Raisonnement Geometrique pour la Programmation des Robots, tesis de Doctorado, Institut National Polytechnique de Grenoble, 1986.

[10] Ramos, O., OF, tesis de Magister, Universidad de los Andes, 1991.

[11] Villegas, E., Ambientes de Programación de Robots, Tesis de Magister, Universidad de Los Andes, 1991.

SOFTWARE as an export product from developing countries: Are there any real chances?

Eduardo Moreira da Costa

TELECOMUNICAÇÕES BRASILEIRAS S/A - TELEBRÁS
Centro de Pesquisa e Desenvolvimento - CPqD
Caixa Postal 1579
13085 -Campinas- SP
Brazil

Índice

1 Introduction 2
2 Market 2
3 Comparative advantages and disadvantages 3
3.1 Talent 3
3.2 Creativity 3
3.3 Small investment 4
3.4 Production technology 4
3.5 Entrepreuneurial interest 4
3.6 Legislation 5
3.7 Specification 6
3.8 Trade mark 6
3.9 Indigenous problems 6
4 What to do? 7
5 Conclusion 8
6 Bibliografia 8

1 Introduction

The world is divided today into two large blocks: those countries who have the latest technologies or have access to them; and those who have not. Exports from the first block are increasing; from the second, are either stable or decreasing. The main comparative advantages of developing countries: cheap labour, natural resources and available spaces and lands [2] are becoming less and less important as the new industries are automated and require less labour, as new materials are developed, and as biotechnology starts to enable production in very short spaces. It is a question of economic survival, and also a question of raising the quality of internal labour positions, that any country be planning what will be its high-technology exports in the years to come [3].

Software is a lesser known product than hardware in the informatics market, albeit being as or perhaps even more important than the latter. The overall market is very large, but in its infancy still in the developing world. Although some individual efforts have already taken place, like the establishment of the International Institute for Software Technology [5] in Macau, within the realm of the United Nations University, overall awareness in the developing world to the importance of software in the world economy today is still very low.

There are some solid and some conjectural reasons that justify the choice of software as a viable export for these countries. The topic is discussed in this article identifying opportunities and difficulties for the establishment of a national and internationally active software industry in developing countries, together with the specific actions to be taken in order to make it happen.

The following sections bring an overview of the full paper that is in preparation.

Section 2 presents some data about the international market for software and its main characteristics. In the following section, the comparative advantages and disadvantages of the developing countries to enter the market are outlined. Section 4 describes actions to be taken in order to establish the industry in the light of the Brazilian experience which might be clicked in other countries. Finally, section 5 concludes the article and presents some comments of a more personal tone referring to the subject.

2 Market

The informatics industry as a whole is already today one of three largest in the world, competing directly with the automotive industry and the tourism industry (not to mention prostitution and drugs, large industries about which there are no trustworthy figures...). The annual rate of increase of the informatics industry though is larger than the other two which will soon transform it into the largest industry in the world!

Software, within the informatics industry has only recently started being accounted for separately. Estimates vary widely both in absolute terms and in the annual rates of increase. It is difficult to have precise figures about the international market for software because there are different interpretations of what is a software product. There is off-the shelf software that is purchased in stores, there is software that is contracted outside a company for specific purposes, there is software that is bundled with hardware, there is software that is developed internally, etc.

Depending upon the classification, it is possible to say that the software industry sells between US$100 and 200 billion a year and grows at a rate between 12 and 30% [4]. These figures do not

take into account software that is developed internally to the companies and not properly accounted for, which is estimated at $250 billion a year, doubling until 1995! [6]

The international market for software as a product is split amongst a very large number of companies of small to medium size. The number of software companies in the US, for instance, is one order of magnitude bigger than that for hardware. The market is generally dominated by US companies, to the order of 60% [4].

The international demand for new software professionals is increasing at such a pace that extrapolations of the present trend towards the year 2000 indicate that all the population of the US, Germany and Japan will be writing software by the end of the century! The way out of such an absurd is clearly a dramatic change in the ways, methods and tools with which software is produced today and, more important to this discussion here, also the entrance of new players in the field of software production.

In summary, the market is very very large. The developed world cannot and will not meet the demand alone. Opportunities are there for new players within the conditions set out in the following sections.

3 Comparative advantages and disadvantages

There are several reasons why developing countries might have a chance in the software market. In the sections that follow a short list of the main reasons is presented.

3.1 Talent

Only a very small percentage of children that enter the educational system in developing countries reach university. Of those who do manage to get there, there is a concentration of talent in the technological areas like engineering and computer science. The reason for this is that these professions carry a high social value in the developing world, much larger than what it gets in the developed world. A worried English father would probably want his brightest son to go to Oxford or Cambridge and then become a civil servant. In the US it would be Harvard (law or perhaps an MBA) and then become a successful lawyer. In our countries it would be a good University (in Brazil, a Federal University or one of the state universities in the state of São Paulo) in an area like engineering, and then become an employee of a large company...

This is to say that the students who attend our best universities in engineering or computer science are of a very good level, even internationally. It is well known that students from the developing world who go abroad for postgraduate studies in these fields do very well in general. Unfortunately some of these do so well that decide to stay on for productive work so badly needed in their home countries, most cases for lack of opportunities back home.

So, talent is available, perhaps not in quantity, but at least in quality.

3.2 Creativity

A good software piece, to work properly, must do what it is supposed to do. That is obvious. But it should do nothing else, and that is not so obvious. To write a good software, the developer must have what is sometimes called lateral thinking: you got to be able to think of all the

unforeseen situations that might occur! In other words, it is important to foresee all the exceptions.

Now, who is capable of thinking about exceptions? Those who are used to it. Who is? Us, for example in Brazil, and that is probably true of many other developing countries. There is a saying in Brazil that goes: "in our country, exception is the rule!".

A good example of this is the report from a bank employee in Brazil describing the misery he went through trying to explain to an international consultant that the brazilian currency in a software program had to be a variable, since it had changed three times in five years!!!

The point here is that surviving decently in many countries involves a lot of creativity: over many decades this has become a trace of local culture that shows itself in many cultural manifestations like music, romance, plays, films, etc., and that could now become very useful in the software market.

3.3 Small investment

Producing software is not cheap! The days of the big software hit produced in a garage using an unreliable 8-bit microcomputer are gone! Professional software is produced today in modern installations, by well organised companies (though small, in general), using networks of high-speed workstations and substantial amounts of software tools.

There is a substantial investment then. What must be emphasised here is that this investment is very small in comparison to what is needed for the production of any other high-technology product. Take, for instance, micro-electronics or chemistry or new materials: the levels of investment to become a player in these fields are huge! Not so for software.

3.4 Production technology

The technology for the production of software is well known and understood: methods, tools, infrastructure, human resources, etc. A company in the developing world can design and implement an infrastructure very much like the one existing at for instance Microsoft in the US with no big problem.

The production technology is therefore non-proprietary in this case which makes it much easier for new players to come into the field.

Another favourable aspect of this item is that "sourcing" is not a problem. If one wants to produce microcomputers, for instance, sourcing for hard-disks, microchips, CRT's, etc., is a major concern. Again, not so for software. Components, if needed, can be bought in the market and transported very cheaply to the "assembly plant".

3.5 Entrepreuneurial interest

In many developing countries a large community of entrepreneurs and businesspeople has grown over the years when the economic model was that of import substitution. As these countries try to follow the international trend towards a more global economy, this sector of the society is eagerly looking for alternatives to their main core business.

This interest can be tapped, with appropriate incentives, for the software sector, since informatics in general is seen as a lucrative sector and has received a lot of media coverage in recent years (in fact in many places owning a microcomputer became much of a status symbol, even if the machine is being used just for very trivial tasks in the office).

3.6 Legislation

Although this question is very different from country to country, it is fair to say that legislation referring to the concession of incentives and special conditions for one particular sector of the economy is easier to get through the whole process of approval. That might be very important for the establishment or the strengthening of the software industry in any country.

There are of course many obstacles to be overcome. Some of them are presented here.

3.7 Specification

It is very difficult to know what the market wants or needs being away from that market. This is probably the biggest obstacle to the export initiative of any product, even more so in this case of specialised products.

One possible way out of this is to get the specifications from a company in the market (an international company) and get the revenues just for the development of the software product (coding). It is not ideal, since most of the revenues will be outside but still it is an entry point [1]. India is taking that road with a very active software industry.

Another initiative would be to buy the way into the market, either buying companies in the developed world or establishing joint-ventures with them for the development and marketing of software products. This is not easy though, and the subject must be very well understood, studied and debated to reach meaningful conclusions.

3.8 Trade mark

It is very common in the developed world to see people buying products by their brand name or make, regardless of where, how or when it was produced. Such is the case in the car industry, for instance. I asked several Canadians and Americans if they new that the car named "FOX" that they bought and used was actually produced in Brazil, and none of them new it. For them it was a Volkswagen car. The place where it was actually produced was irrelevant.

That is the power of the trade mark name. In order to be successful in the export market, a country would need one or a few brand names that were familiar to the public. That costs a lot of money to build and maintain but is a problem that cannot be overlooked.

3.9 Indigenous problems

Being in the developing world, with all the problems related to infrastructure, political organisation, unstable governments, etc., a company will be faced with serious problems to establish itself in the international arena.

Quality is a fundamental characteristic in the software business and that culture must be enforced in the industry. There is no other way. If the effort to establish a trade-mark referred to above is not paralleled by an effort to improve quality of products, in a general sense, the first effort will be useless.

Another big problem is that a major effort in any industry will need definitive and articulate government approval and support. The governments of developing worlds in general lack credibility and might jeopardise any serious effort. In some countries the government involvement might even become the "kiss of death": if they are in, it is not worth the effort and energy...

Lack of communications infrastructure and of access to information are other two major problems. Both have to be dealt with and resolved before anything serious can take place.

4 What to do?

Here the problem has to be individually addressed since solutions will differ from country to country. Some aspects of the problem can be considered general though, in the sense that every country will have to face them:

A. Planning

i. Studies - What is the international market, opportunities, niche markets, specific demands, channels of distribution, etc. Situation in the country: existing companies, potential for export, existing products, geographical distribution, etc.

ii. Short-term planning - What to do to convince the society that a plan is under way and it is for real. Funding for the initial steps in the plan. Establishment of national goals to be pursued, etc.

iii. Coordination of efforts - Definition of actors and rôles and what is expected of them in the national plan. Coordination of these efforts towards the common objective.

iv. General plan - Establishment of definite objectives, goals and means to achieve them with a 10 year horizon, stating clearly the responsibilities for each step of the plan.

B. The model company

i. The business - Business plan for the company. Demand, production, delivery, revenues, financial resources, location, trade-mark, investment, return on investment, etc.

ii. Infrastructure - Hardware and software needs. Networking. Communications, internal and external. Plant, specific building needs, working environment.

iii. Human resources - Number of people. Qualifications. Training.

iv. Funding - Special schemes for the star-up company. Government funding, venture capital, own investment, incentives for spin-offs from large companies, etc.

C. The technopolis

i. Existing sites: Places (towns, regions) in the country where there is already some effort under way to establish a concentration of high-tech industries.

ii. Software technopolis : Prerequisites for participation in the plan, choices of initial places.

iii. Pilot experience : For each chosen site, infrastructure, coordination with local governments, incentives, follow-up and conclusion

5 Conclusion

The world of the future will be dominated by the countries who detain or have access to the latest technologies. This brings several concerns for developing countries. Discussions on the subject are lengthy and endless. I would like to concentrate on one aspect of the problem: that of quality of labour.

High-technology labour posts are in general clean (not always safe though), climatized, with the latest and more comfortable furniture, in well-developed regions of a country, close to the best educational institutions, and, most important, well paid. The reader might have noticed that I did not mention in the advantages of the developing countries for the production of software the fact that programmer's salaries are one or two orders of magnitude lower. And that was not a lapse: this industry should be established in our countries to raise the standards of labour and living of our citizens *including getting paid on an international level.* At first the lower salaries will be an advantage, of course. But the industry should not be created based upon this assumption. It should be competitive because of the other advantages listed in section 3 and even more so at the beginning because of the lower salaries.

Another opportunity that was not mentioned previously is that we should look for niche markets as is sometimes recommended by specialists. I believe that we should indeed look at niche markets as one segment of the general market. But we should look at the mainstream markets as well and think big: no inferiority complex should blur our vision and diminish the scope of the programmes.

I sincerely believe that the chance is there for the taking. Brazil is going ahead with such a programme [7] and is willing to share its initial studies with everyone. This article is just an overview of the subject and I welcome any comments, questions or requests for information that the interested reader might want to address me. I would like to acknowledge fruitful discussions with and contributions from A. Salles, F. Souza, C. Violato, C. Mammana, V. Almeida, T. Takahashi, I. Campos, S. Meira, A. Moura, C. Lucena, E. Araújo, G. Lucena and J. Portugal, who helped me stay firm on the ground when enthusiasm pushed me too far out on expectations...

6 Bibliografia

[1] Dertouzos, M.L.: "Communications, computers and networks", Scientific American, v. 265, n. 3, Sept. 1991, pp. 30-37.

[2] Colombo, U.: "The technological revolution and the future of the third world", IEEE Technology & Society, v. 10, n. 1, Spring 1991, pp. 25-32.

[3] Schneider, H-J.: "Software production: organization and modalities", UNIDO, Vienna, Austria, IPCT.63, 5/17/88, 76 pp..

[4] Brandt, R. et allii: "Can the US stay ahead in software?", Int'l Business Week, March/11/1991, pp. 62-67.

[5] Bjorner, D.: "The United Nations University International Institute for software technology",xxx,1992

[6] Sommerville, I.: "Software Engineering" - 3rd. Ed., Addison-Wesley, Wokingham, UK, 1989, 653 pp..

[7] "Strategic Development of Informatics in Brazil: Coordination of efforts and definition of new directions", Poject Document CNPq-UNDP, Brasília, Brazil, 1992, 356 pp.

IMPLEMENTATION OF AN ARTIFICIAL NEURAL NETWORK FOR THE SEGMENTATION OF TEXTURE IMAGES.

A. Mosquera, D. Cabello, M. J. Carreira and M. G. Penedo.
Departamento de Electrónica y Computación. Facultad de Física.
Universidad de Santiago de Compostela.
15706 Santiago de Compostela. SPAIN.

Abstract: In this work we describe the implementation of an artificial neural network, an extension of Hopfield's model, for the segmentation of textured images. We use a Markov random field in order to model the textures in the image. The problem is approached in terms of the minimization of a cost function that is projected onto the network. It provides a locally optimal solution to the problem of the classification of M*M pixels into K classes (textures). The experimental results obtained on artificial and natural images show the validity of the architecture we propose.

Keywords: *Image analysis, texture modeling, Markov random field, textured image segmentation, artificial neural network, Hopfield network.*

1. INTRODUCTION.

Image segmentation is an important problem in artificial vision, image analysis, etc. It consists in the partition of an image into a set of elementary regions characterized by the fact that some property is constant. Many objects in the real world present textured surfaces; the information texture provides is determinant in order to obtain a correct segmentation of the image. There are currently a large number of segmentation methods based on statistics obtained from the values of the gray levels of the pixels in the image. In our approach we will use the concept of Markov random field as a model for the image textures, [1,2].

The inherent parallelism that both the problems of image processing and neural network architectures present have increased the interest in their use for the solution of difficult computational problems, such as the segmentation problem, [3,4]. In most cases, these neural networks are designed so as to minimize an energy function given by the architecture of the network itself, [5]. Its parameters are thus obtained from the cost function

we want to minimize for the solution of the problem. In practical cases, the networks must have the lowest possible number of connections. In this context, Markov's random field plays an important role as it limits the dependencies of each pixel to a small set of neighbors.

In this work we describe the implementation of an artificial neural network for the segmentation of textured images which is an extension of Hopfield's model [5]. The problem is approached in terms of the minimization of a cost function that is projected onto the network. This function provides a locally optimal solution for the problem of the classification of M*M pixels into K classes (textures). The results obtained, both on artificial images, a visualization of Markov's field, and on natural images extracted from Brodatz's album [6] prove the validity of the architecture we propose.

2. IMAGE MODEL.

The use of Markov's random field model in image processing applications has been investigated by many authors, and in the literature there is a detailed discussion of its use in images where textures are present, [1,2,7]. In this work we will use a second order Markov random field in order to model the probability density of the image intensities matrix.

In a Markov field model, the intensity of the pixels of an image is a linear combination of the intensities of neighboring pixels. We denote as Ω the set of points in the image of dimension M*M, $\Omega = \{ (i,j) ; 0 \leq i,j \leq M-1 \}$, and s is a two dimensional vector such that $s \in \Omega$. If we assume that the observations y(s) of a texture are Gaussian and have a zero mean value, Markov's random field model can be described by means of the following equation:

$$y(s) = \sum_{r \in N_s} \theta_r \, y(s+r) + e(s) \qquad (1)$$

where e(s) is a sequence of Gaussian noise with a null mean value, N_s is the neighbor vector considered and θ_r is the parameter vector of the model. This implies that:

$$p(y(s) / y(r), r \neq s) = p(y(s) / y(s+r), r \in N_s) \qquad (2)$$

That is, $\{y(\cdot)\}$ is strictly a Markov field with respect to its neighbors.

The unknown parameters $\{\theta,\upsilon\}$ can be estimated for each pixel of a given image by means of the equations

$$\theta^* = \left[\sum_{\Omega} q(s)q^t(s)\right]^{-1}\left[\sum_{\Omega} q(s)y(s)\right]$$
$$v^* = \frac{1}{M^2}\sum_{\Omega}\left[y(s)-\theta^{*t}q(s)\right]^2 \qquad (3)$$

where q(s) is the neighbor vector of y(s) and the sums extend to parameter estimation windows of size Ω'. These equations constitute a maximum probability estimation for the parameters of the model, [8].

3. NEURAL NETWORK FOR TEXTURE CLASSIFICATION.

Once the image model has been established and its parameters estimated in each position using (3), we will approach the segmentation problem by means of the implementation of an artificial neural network which minimizes an energy function obtained from the parameters given by the image model we consider.

The neural network we use is made up of K layers, each one of them consists in a matrix of M*M neurones, where K is the number of possible textures existing in the image and M the dimensionality of the image. We assume that the elements (neurones) of the network are binary and are indexed as (i,j,l), where (i,j) indicates the position in the image and l the layer. The (i,j,l)-th neurone is ON if its output V_{ijl} is 1, and this will indicate that pixel (i,j) of the image belongs to texture l, assuming that each pixel has a single label. Let $T_{ijl:i'j'l'}$ be the weight associated with the interconnection between neurones (i,j,l) and (i',j',l') and I_{ijl} the input of neurone (i,j,l) and let us use a network with symmetric connections, $T_{ijl:i'j'l'}=T_{i'j'l':ijl}$, and without feedback, that is $T_{ijl:ijl}=0$. Then, the general expression of the energy of the network is:

$$E=-\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{l=1}^{K}\sum_{i'=1}^{M}\sum_{j'=1}^{M}\sum_{l'=1}^{K} T_{ijl:i'j'l'}V_{ijl}V_{i'j'l'}-\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{l=1}^{K} I_{ijl}Vijl \qquad (4)$$

The use of this network means obtaining a set of states of the neurones so that its energy is a minimum. In order to employ the spontaneous process for the minimization of the energy of the neural network, we will formulate the segmentation problem as a minimization of an error function which is the sum of two terms. The first term corresponds to a quadratic

error of the assignment of an image point to a given texture and the second favors the assignment of neighboring points to the same texture. This error function is given by

$$e=\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{l=1}^{K}\left[\theta_{ij}-\hat{\theta}_l\right]^2 V_{ijl}-\frac{\lambda}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{l=1}^{K}\sum_{(i',j')\in N_s} V_{ijl}V_{i'j'l} \qquad (5)$$

where θ_{ij} is the M*M matrix of parameter vectors estimated using (3), $\hat{\theta}_l$ is the characteristic parameter vector of each layer (texture), N_s is the neighbor vector considered and λ is a weight factor.

From equations (4) and (5) we can identify the parameters of the network, these parameters will be

$$I_{ijl}=-[\theta ij-\hat{\theta}_l]^2$$

$$T_{ijl:i'j'l'}=\begin{cases}\lambda & \text{if } (i',j')\in N_s \text{ and } l'=l \\ 0 & \text{other case}\end{cases} \qquad (6)$$

We have implemented a relaxation algorithm in order to minimize the energy of the network. This energy can be written as

$$E=\sum_{i=1,j=1,l=1}^{M,M,K} E_{ijl}=\frac{-1}{2}\sum_{i=1,j=1,l=1}^{M,M,K} U_{ijl}\,V_{ijl} \qquad (7)$$

where U_{ijl} is the potential of the (i,j,l)-th neurone. By identifying terms with expression (4), we can obtain the expression of the potential of a neurone, which is given by

$$U_{ijl}=\sum_{i'=1,j'=1,l'=1}^{M,M,K} T_{ijl:i'j'l'}\,V_{i'j'l'}+I_{ijl} \qquad (8)$$

For our network and taking into account the values of its parameters, equation (6) is

$$U_{ijl}=\lambda\sum_{(i',j')\in N_s} V_{i'j'l}-(\theta_{ijl}-\hat{\theta}_l)^2 \qquad (9)$$

In the relaxation algorithm, in order to generate a new state of the neural network, we assume that each pixel has a single label, which means that only one neurone is active in each column. The new state of the network is calculated by means of the "Winner-takes-all" method over each column so that the neurone with the highest potential is turned ON and the rest are OFF. This process is assumed as simultaneous for all the columns of the network. This scheme makes the system converge to a stable state, generally a local minimum of the function we want to minimize.

Before applying the network to a general image, it is necessary for it to know what textures it may find, that is, it is necessary for it to undergo a learning process. This process consists in the calculation of the Markov field parameters characterizing the texture. These parameters are going to be the ones characterizing each layer of the neural network.

3. RESULTS.

In order to test the validity of the artificial neural network and of the relaxation algorithm we propose, we have generated images which are a composition of different textures and attempted their segmentation. Figure 1.a shows a composition of textures which were artificially generated as a visual representation of Markov random fields with different values of the parameters. Figure 1.b, on the other hand, shows a composition of natural textures (beach sand, water and wood grain) extracted from Brodtaz's album [6]. Both images have 256*256 pixels, 256 levels of gray and present a combination of three different textures; the network necessary in order to segment them will have three layers. The parameters $\hat{\theta}_l$ characterizing each texture/layer of the network are estimated from 25*25 pixel samples for the textures of figure 1.a and from 49*49 pixel samples for those of figure 1.b.

An important aspect in the segmentation process is the determination of the optimum window size Ω' for the estimation, using (3), of the parameter vector θ_{ij} (local properties). This window must be large enough for the estimation of the parameters to be correct and small enough for us to be able to detect small regions of the image. Its size will be determined automatically by means of the analysis of the codifference matrices obtained as a generalization of the operation of binary images erosion to multiple gray level images using a structural element. The distance between the minima of the codifference matrices sum function provides information about the size of the 'motif' of the texture, [9]. This method fixed a window size Ω' of 9*9 pixels for figure 1.a and of 17*25 for figure 1.b. Also, in this last case, we have generalized the concept of neighborhood, taking as closest neighbors to a

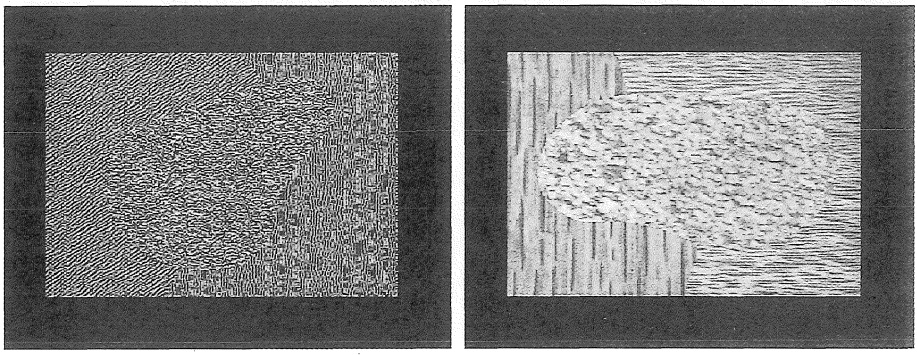


Figure 1. Texture composition images. (a): artificially generated textures, (b): natural textures.

given pixel those located at a distance of half the size of the motif. This implies that in the second term of equation (5), N_s includes the neighbors we consider and all the pixels inside them.

The value of parameter λ is not critical in the segmentation process. Within a small interval, higher values lead to more compact segmentations; outside this interval, its value does not affect the results obtained very much. In the examples considered we have taken values in the interval (1,3).

The results obtained when applying the network to the image of figure 1.a considering $\lambda = 1$, starting from a random distribution of neurone activation and after 40 iterations are shown in figure 2; in figure 2.a the regions obtained appear as masks with different levels of gray and in figure 2.b, we show their contours superimposed on the original image. As we can see, the boundaries obtained coincide with those perceived on the original image.

Figure 3 shows the segmentation obtained for the image of figure 1.b. From left to right and from top to bottom, figures 3.a and 3.b show the results provided by the network and the deterministic relaxation algorithm considered, $\lambda = 2$, starting again from a random distribution of the activation of the neurones and after 40 iterations. In order to compare, figures 3.b and 3.c show the segmentation provided by a minimum distance classifier, taking into account three classes whose characteristic patterns are parameters $\hat{\theta}_i$, assigning to each pixel the class (texture) which is closest in the property space and transferring this information to the image space. As we can observe, the network presents a better performance, reducing segmentation errors. We can conclude by pointing out the validity of

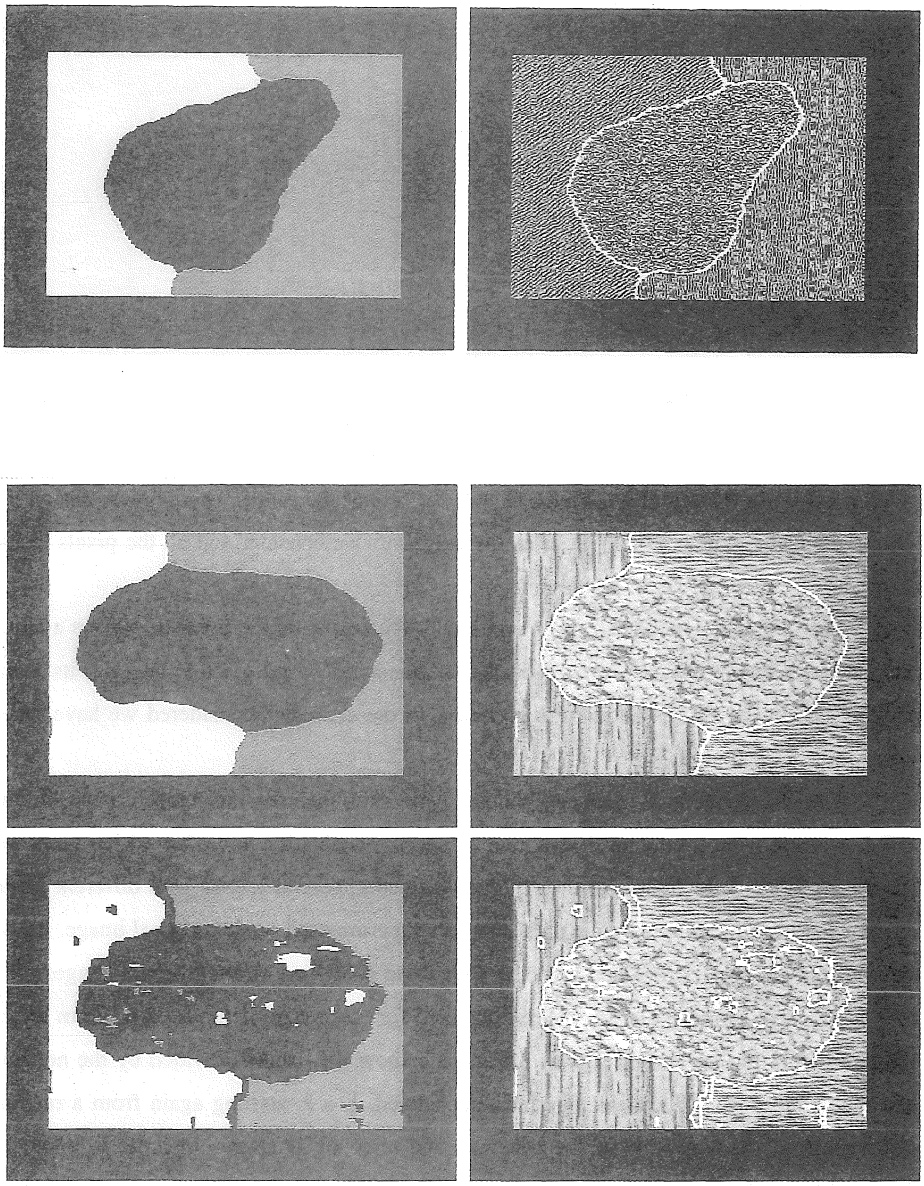


Figure 3. Segmentation of natural textures. (a),(b): results obtained using the neural network, (c),(d): results obtained using ca minimum distance classificer.

the architecture we propose, which provides almost optimal results for the problem of the segmentation of texture images.

ACKNOWLEDGEMENTS.

This work was supported by Spanish CICYT, TIC91-0816, and Xunta de Galicia, XUGA20602A90.

REFERENCES.

[1]. M. Hassner and J. Sklansky. "The use of Markov random fields as model of texture". Comput. Graphics Image Processing, vol. 12, pp. 357-370, 1980.

[2]. G.R. Cross and A.K. Jain. "Markov random field texture models". IEEE Trans. Pattern. Anal. Machine Intell., vol. PAMI-5, pp. 25-39, 1983.

[3]. B.S. Manjunath, Tal Simchony and R. Chellappa. "Stochastic and deterministic networks for texture segmentation". IEEE Trans. Acoust., Speech, Signal Process. vol. 38, pp.1039-1049, 1990.

[4]. Chung-Lin Huang. "Parallel image segmentation usin modofied Hopfield model". Pattern Recognition Letters vol. 13, pp.345-353, 1992.

[5]. J.J. Hopfield and D. W. Tank. "Computing with Neural Circuits: A model". Science, vol. 233, pp.625-633, 1986.

[6]. P. Brodatz. "Textures: A photographic album for artistis & designers". Dover Publications, New York, 1966.

[7]. R. Chellappa. "Two-dimensional discrete Gaussian Markov random field models for image processing", in *Progress in Pattern Recognition 2* (L.N Kanal and A. Rosenfeld, Eds.), pp. 79-112, Elsevier, New York, 1985.

[8]. R.L. Kashyap and R. Chellappa. "Estimation and Choice of Neighbors in Spatial-Interaction Model of Images". IEEE Trans. Information Theory, vol. IT-29, pp. 60-72, 1983.

[9]. A. Mosquera, D. Cabello, M.J. Carreira and M.G. Penedo. "Unsupervised textured image segmentation using Markov random field and clustering algorithms". In *Computer Analysis of Images and Patterns* (R. Klette, Ed.). Research in Informatics, vol. 5, pp.139-147, Akademie Verlag, Berlín, 1991.

Sistemas de Tableau com Unificação para Lógicas Modais[†]

Marcos Mota do Carmo Costa
Professor conveniado do DI-UFPE
Pesquisador da EMBRAPA
SHCE 711 E 302
70660 Brasilia DF Brasil

Sumário

O método dos tableaux tem se mostrado bastante intuitivo embora ineficiente, em virtude da dificuldade em se controlar as instanciações em lógicas de 1ª ordem. Estudos recentes comprovaram a eficiência de provadores automáticos de teoremas baseados em tableau com a incorporação do algoritmo de unificação para a lógica clássica de 1ª ordem.

Neste artigo propomos um sistema de tableau para as lógicas modais K, T, D, K4 e S4 com a fórmula de Barcan utilizando o algoritmo de unificação no controle das instanciações.

Palavras chaves: Provadores automáticos de teoremas, Sistema de tableau, Lógica modal e Algoritmo de unificação.

1 Introdução

Após [12] e até meados da década de 80, o sistema de tableau foi evitado como base para provadores automáticos de teoremas em virtude de sua ineficiência causada por inúmeras instanciações de variáveis dificilmente direcionáveis. Estas instanciações puderam ser evitadas—ou melhor, direcionadas—por Reeves [11], com a incorporação do algoritmo de unificação (*cf.* [12]) no sistema de tableau.

Estudos recentes vieram dar suporte a deduções mecanizadas para as lógicas modais. Ressaltamos [9], com o método de resolução, [14] com o método de matriz com conexão e [5, 2] com o método de tableau com unificação.

Neste artigo propomos um sistema de tableau para as lógicas modais K, T, D, K4 e S4 com a fórmula de Barcan utilizando o algoritmo de unificação no controle das instanciações.

Na seção 2 os sistemas de tableaux para as lógicas clássica e modal de 1ª ordem são apresentados sob forma resumida. Na seção 3 é apresentado o sistema de tableau com unificação para as lógicas modais consideradas. Conclusões são tecidas na seção 4.

É assumido que o leitor esteja familiarizado com as lógicas clássica e modal, com o método de resolução e tenha ao menos uma noção inicial a respeito do sistema de tableau.

[†]Pesquisa parcialmente financiada pelo CNPq e FACEPE.

2 Sistemas de Tableaux

O método dos Tableaux de prova refutacional para a lógica clássica de 1ª ordem é devido aos trabalhos de Hintikka [8] e Beth [1] e apresentado de forma elegante, sob uma notação uniforme por Smullyan em [13].

Nesta seção iremos apresentar o sistema de tableau inicialmente para a lógica clássica e, posteriormente, para os sistemas K, T, D, K4 e S4 da lógica modal de 1ª conforme descrito em [4].

2.1 Sistema de Tableau para a Lógica Clássica

Regras Tipo A:	$\alpha \wedge \beta$	$\neg(\alpha \vee \beta)$	$\neg(\alpha \Rightarrow \beta)$	$\neg\neg\alpha$
	α	$\neg\alpha$	$\neg\alpha$	α
	β	$\neg\beta$	$\neg\beta$	
Regras Tipo B:	$\alpha \vee \beta$	$\alpha \Rightarrow \beta$	$\neg(\alpha \wedge \beta)$	
	$\alpha \mid \beta$	$\neg\alpha \mid \beta$	$\neg\alpha \mid \neg\beta$	
Regras Tipo C:	$\forall x\alpha$	$\neg\exists x\alpha$		
	$\forall x\alpha$	$\neg\exists x\alpha$		
	$\alpha(x/t)$	$\neg\alpha(x/t)$,	onde t é um termo qualquer	
Regras Tipo D:	$\exists x\alpha$	$\neg\forall x\alpha$		
	$\alpha(x/c)$	$\neg\alpha(x/c)$,	onde c é uma constante nova para o tableau	

2.2 Sistema de Tableau para Lógica Modal com a Fórmula de Barcan

Uma refutação baseada em tableau para lógica modal requer o uso de diversos sub-tableaux. A criação e manipulação destes sub-tableaux vem, de uma certa forma, simular as mudanças de mundo em uma estruturade mundos possíveis de Kripke (*cf.* [10]). Se a fórmula de Barcan $(\forall x \Box\alpha \Rightarrow \Box\forall x\alpha)$ é aceita como axioma, então as regras do sistema implicam na adição de fórmulas a (sub-)tableaux já existentes. Portanto, faz-se necessário definir um aparato para identificação dos tableaux e suas fórmulas, o que é feito a seguir.

Definição 2.1 *Um* prefixo *é qualquer expressão utilizada para dar nome a um tableau que pode aparecer na refutação por tableau de uma dada fórmula.*

A idéia é ter um nome diferente para cada tableau de uma refutação. Então, uma fórmula α em uma refutação por tableau é unicamente identificada pelo par $(\mathcal{T}, \alpha)$, onde $\mathcal{T}$ é o nome do tableau onde α ocorre.

Para administrar a criação de novos tableaux e adição de fórmulas a tableaux existentes, é criado o operador ρ que aplicado a um prefixo $\mathcal{T}$ e uma fórmula α produz um dos dois resultados abaixo:

- Cria um novo tableau, cujo nome é $\mathcal{T}$, começando com α em sua forma skolemizada e registrando os quantificadores do tableau original, se $\mathcal{T}$ não é um nome de algum tableau existente.
- Adiciona α ao tableau designado pelo prefixo $\mathcal{T}$, caso contrário.

Apresentamos a seguir, o sistema de tableau para a lógica modal de 1ª ordem, que é formado pelas regras do sistema de tableau para a lógica clássica proposicional acrescido das regras específicas para cada sistema da lógica modal (onde α e β são fórmulas):

Sistema K: Formado pelas regras da lógica clássica mais as seguintes regras:

Regras Tipo E:

$$\frac{\Box\alpha}{\rho(\mathcal{T}',\alpha)} \qquad \frac{\neg\Diamond\alpha}{\rho(\mathcal{T}',\neg\alpha),}$$

Onde $\mathcal{T}'$ é um tableau previamente gerado por aplicação da regra F a uma fórmula do ramo.

Regras Tipo F:

$$\frac{\Diamond\alpha}{\rho(\mathcal{T}',\alpha)} \qquad \frac{\neg\Box\alpha}{\rho(\mathcal{T}',\neg\alpha),}$$

onde $\mathcal{T}'$ é um novo tableau.

Sistema T: Formado pelas regras do sistema K mais a seguinte regra:

Regras Tipo E-R[1]:

$$\frac{\Box\alpha}{\alpha} \qquad \frac{\neg\Diamond\alpha}{\neg\alpha}$$

Sistema D: Formado pelas regras do sistema K, sendo que a restrição aplicada à regra E é modificada para:

$\mathcal{T}'$ é um novo tableau ou um tableau previamente gerado por aplicação da regra E ou F a uma fórmula do ramo.

Sistema K4: Formado pelas regras do sistema K mais a seguinte regra:

[1] "R" de reflexiva.

Regras Tipo E-T[4]:

$$\frac{\Box\alpha}{\rho(\mathcal{T}', \Box\alpha)} \qquad \frac{\neg\Diamond\alpha}{\rho(\mathcal{T}', \neg\Diamond\alpha),}$$

Onde $\mathcal{T}'$ é um tableau previamente gerado por aplicação da regra F a uma fórmula do ramo.

Sistema S4: Formado pelas regras dos sitemas T e K4.

Observação: O leitor é convidado a verificar que a regra E pode ser eliminada do conjunto de regras de tableau para o sistema S4, por ser redundante.

[Fim_da_Obs]

3 Sistemas de Tableaux com Unificação

Nesta seção apresentamos os sistemas de tableaux com unificação para a lógica clássica e, posteriormente, para a lógicas modais sendo consideradas.

3.1 Tableau com Unificação para a Lógica Clássica

O sistema de tableau com unificação para fórmulas skolemizadas da lógica clássica de 1ª ordem pode ser definido através das regras A e B do sistema sem unificação mais a regra C abaixo e a incorporação do algoritmo de unificação na obtenção de pares de fórmulas atômicas complementares.

Regras Tipo C:

$$\frac{\forall x\alpha}{\forall x\alpha} \qquad \frac{\neg\exists x\alpha}{\neg\exists x\alpha}$$

$$\alpha(x/y) \qquad \neg\alpha(x/y),$$

onde a variável y é nova para o tableau

3.2 Sistema de Tableau com Unificação para Lógica Modal

Iniciamos a apresentação do sistema com algumas definições necessárias para obtenção da forma normal de Skolem de uma fórmula dada.

As formas normais de Prenex e Skolem são obtidas como na lógica clássica, considerando somente os operadores externos aos operadores modais. Para exemplificar, consideremos a fórmula

$$\forall x\Box\exists y P(x,y) \Rightarrow \forall z\Box R(z)$$

que tem

$$\forall z(\Box\exists y P(f(z),y) \Rightarrow \Box R(z)$$

[4] "T" de transitiva.

como uma forma de suas formas normais de Skolem.

Definição 3.1 *A ocorrência de uma subfórmula φ tem uma polaridade positiva em uma fórmula se α está no escopo de um número par de "$\neg$", implícitos ou explícitos. Caso contrário, α é dita ter uma polaridade negativa.*

As regras do sistema de tableau com unificação são derivadas das regras do sistema sem unificação com as seguintes modificações:

- Em qualquer estado do tableau, qualquer fórmula está na forma normal de Skolem (conforme definido acima).
- O algoritmo de unificação é usado a fim de se obter pares complementares de fórmulas atômicas.
- Depois de cada aplicação das regras E ou F, precisamos skolemizar a fórmula resultante, tomando os seguintes cuidados:
  - Ao começarmos um novo tableau com a fórmula α, precisamos skolemizar α considerando os quantificadores do tableau original como se eles fossem os quantificadores mais externos de α;
  - Ao adicionarmos α a um tableau $\mathcal{T}$ existente, precisamos skolemizar α considerando as variáveis quantificadas da fórmula que originou $\mathcal{T}$ como se elas fossem os quantificadores mais externos de α.
- Por causa do comentário anterior, a regra C precisa ser modificada a fim de mantermos a informação dos quantificadores fora das modalidades. Assim, quando aplicamos a regra C, a variável quantificada é anotada no operador modal. Como um exemplo, vamos considerar a fórmula:

  $$\forall x \Box P(x).$$

  Após a aplicação da regra C, obtemos:

  $$\Box^x P(x).$$

  A formalização da regra tipo C é apresentada abaixo.
- O processo de renomeação de variáveis para unificação fica agora sujeito à seguinte restrição: Seja $\Diamond\gamma$ (ou $\neg\Box\gamma$) uma subfórmula de $\forall x\varphi$. Se aplicarmos a regra F a $\Diamond\gamma$ (ou $\neg\Box\gamma$), então não podemos renomear a variável x quando formos unificar a fórmula γ (ou alguma

de suas subfórmulas) no subtableau gerado. E, indicamos este procedimento colocando uma marca "$\bullet$" antes da variável x.

Esta restrição se deve ao fato de que podemos aplicar a regra E mais de uma vez para a mesma fórmula, criando mais de uma instância de uma variável quantificada fora do operador modal e cada vez que aplicamos a regra F, geramos um novo tableau e, portanto, não podemos mais gerar outras intâncias desta variável para este subtableau. O tratamento das variáveis com esta marca é dado a seguir.

- Sejam α uma fórmula e x uma variável ou uma *variável marcada* na forma $x = \bullet y$. Então, a fórmula α^x é obtida pelas regras:
  - $\alpha^x = \alpha$, se α é uma fórmula atômica;
  - $\alpha^x = \neg\beta^x$, se $\alpha = \neg\beta$;
  - $\alpha^x = \beta^x \vee \theta^x$, se $\alpha = \beta \vee \theta$;
  - $\alpha^x = \forall x\beta^x$, se $\alpha = \forall x\beta$;
  - $\alpha^x = \Box^x\beta^x$, se $\alpha = \Box\beta$ com polaridade positiva;
  - $\alpha^x = \Box^{\bullet x}\beta^{\bullet x}$, se $\alpha = \Box\beta$ com polaridade negativa.
- A regra tipo C é agora modificada para:

Regras Tipo C:

$$\frac{\forall x\alpha}{\begin{array}{c}\forall x\alpha \\ \alpha^y(x/y)\end{array}} \qquad \frac{\neg\exists x\alpha}{\begin{array}{c}\neg\exists x\alpha \\ \neg\alpha^y(x/y),\end{array}}$$

Onde a variável y é nova para o tableau e α^y é obtida conforme definido acima.

Teorema 3.1 *Seja α uma fórmula na forma normal de Skolem. Podemos obter um tableau atomicamente fechado para α usando o método dos tableaux sem unificação se e somente se pudermos obter um tableau atomicamente fechado para α usando o método dos tableaux com unificação.*

Prova Omitimos a prova por questão de espaço. Para tanto, o leitor poderá referir-se a [4].
[Fim_da_Prova]

Exemplo 3.1 Vamos provar a seguinte instância da fórmula de Barcan:[3]

$$\forall z\Box P(z) \Rightarrow \Box\forall x P(x).$$

[3] Note que utilizamos prefixos para nomear e distingüir os tableaux.

1. $\mathcal{T}_1$ $\forall z\neg(\Box P(z) \Rightarrow \Box\forall x P(x))$ (forma normal de Skolem fórm. neg.)
2. $\mathcal{T}_1$ $\neg(\Box^y P(y) \Rightarrow \Box^{\bullet y}\forall x P(x))$ (de 1, pela regra C)
3. $\mathcal{T}_1$ $\Box^y P(y)$ (de 2, pela regra A)
4. $\mathcal{T}_1$ $\neg\Box^{\bullet y}\forall x P(x)$ (de 2, pela regra A)
5. $\mathcal{T}_2$ $\neg P(f(\bullet y))$ (de 4, pela regra F e skolemização)
6. $\mathcal{T}_2$ $P(y)$ (de 3, pela regra E)

Assim, o tableau é fechado unificando-se as fórmulas 5 e 6. Note que não podemos renomear a variável $\bullet y$ da fórmula 5, mas podemos renomear a variável y da fórmula 6 a fim de fazermos a unificação.

Exemplo 3.2 Vamos tentar provar a seguinte fórmula:

$$\Box\exists x P(x) \Rightarrow \exists z\Box P(z)$$

1. $\mathcal{T}_1$ $\forall z\neg(\Box\exists x P(x) \Rightarrow \Box P(z))$ (forma normal de Skolem da fórm. neg.)
2. $\mathcal{T}_1$ $\neg(\Box^y\exists P(x) \Rightarrow \Box^{\bullet y} P(\bullet y))$ (de 1, pela regra C)
3. $\mathcal{T}_1$ $\Box^y\exists x P(x)$ (de 2, pela regra A)
4. $\mathcal{T}_1$ $\neg\Box^{\bullet y} P(\bullet y)$ (de 2, pela regra A)
5. $\mathcal{T}_2$ $\neg P(\bullet y)$ (de 4, pela regra F)
6. $\mathcal{T}_2$ $P(f(\bullet y))$ (de 3, pela regra E e skolemização)

Note que a variável $\bullet y$ da fórmula $\neg\Box^{\bullet y} P(\bullet y)$ que originou o tableau $\mathcal{T}_2$ foi considerada na skolemização de $\exists x P(x)$ como se $\bullet y$ fosse sua variável quantificada mais externa, dando origem à fórmula $P(f(\bullet y))$. Agora, não podemos fechar o tableau unificando as fórmulas 5 e 6, pois não podemos renomear a variável $\bullet y$.

4 Conclusão

O refinamento linear para o método de resolução para lógica modal proposicional de Fariñas del Cerro [6] foi proposto em [7]. O refinamento linear com cláusulas ordenadas (OL-deduções) dos métodos de resolução[4] e tableau para lógica modal de 1ª com a fórmula de Barcan foi apresentado em [3]. Por ser bastante intuitivo e permitir uma boa visualisação das provas, o uso do sistema de tableau facilitou sobremaneira na descoberta desta estratégia. Foram também estabelecidas fortes ligações do método sendo apresentado com o princípio de resolução, através um algoritmo de transformação de provas pelo método dos tableaux para resolução e vice-versa que será objeto de um próximo artigo em elaboração.

[4] Método de resolução similar ao de [9]

Referências

[1] E. W. Beth. *The foundations of Mathematics.* North Holland, 1959.

[2] M. M. do C. Costa. *Characterization of Modal [Action] Logic.* PhD thesis, Imperial College, Londres, 1990.

[3] M. M. do C. Costa. Estratégia linear para provadores automáticos de teoremas para lógicas modais. In *VIII Simpósio Brasileiro de Inteligência Artificial*, Brasilia - DF, Brasil, 1991. Sociedade Brasileira de Computação.

[4] M. M. do C. Costa. *Introdução à Lógica Modal Aplicada à Computação.* VIII Escola de Computação. Sociedade Brasileira de Computação, 1992.

[5] J. Cunningham and M. M. do C. Costa. Mechanised deduction and modal action logic. Technical Report R5, UK Alvey Software Engineering Directorate Forest (GEC-Marconi Research Limited), Imperial College, Londres, 1988.

[6] L Fariñas del Cerro. Resolution modal logics. *Logique et Analyse*, 1986.

[7] L. Fariñas del Cerro. Linear modal deductions. In *CADE 9*, 1988.

[8] J. Hintikka. Form and content in quantitification theory. *Acta Philosophica Fennica*, 8, 1955.

[9] K. Konolige. Resolution and quantified epistemic logics. In *Proc. of CADE-8*, 1986.

[10] S. Kripke. A completeness theorem in modal logic. *Journal of Symbolic Logic*, 24(1), march 1959.

[11] S. V. Reeves. *Theorem-proving by Semantic Tableaux.* PhD thesis, University of Birmingham, 1985.

[12] J. A. Robinson. A machine-oriented based on the resolution principle. *JACM*, 1965.

[13] R. M. Smullyan. *First-Order Logic.* Springer-Verlag, Berlim, 1968.

[14] L. A. Wallen. *Automated Proof Search in Non-Classical Logics: Efficient Matrix Proof Methods for Modal and Intuitionistic Logics.* PhD thesis, University of Edinburgh, 1987.

Um Ambiente de Resolução de Problemas para Sistemas Baseados em Conhecimento

Alessandro Mueller (*)
Fernanda dos Santos Cunha (*)
Leandro José Komosinski (**)

Departamento de Ciências Estatísticas e da Computação (DCEC)
Centro Tecnológico
Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476
Florianópolis - SC
e-mail: cec1ljk@brufsc.bitnet

RESUMO

Neste artigo é apresentado um ambiente de resolução de problemas híbrido, baseado na combinação dos formalismos de representação do conhecimento de Regras de Produção e Frames. Cada formalismo é utilizado para representar uma parte do problema. Aplicações que utilizam mais de um formalismo podem ter desempenho superior aos sistemas de um único formalismo.

palavras-chaves: sistemas baseados em conhecimento, sistemas híbridos, frames e regras de produção

ABSTRACT

In this paper we show an Knowledge-based environment for problem solving. The environment combines two formalisms for knowledge representation: production rules and frames. Each formalism models a specific part of knowledge domain. Applications based on both formalisms can have better performance than others based on only one.

key-words: knowledge-based systems, hybrid systems, frames and production rules

* Alunos do Curso de Bacharelado em Ciências da Computação - UFSC
** Professor (MSc) do Curso de Bacharelado em Ciências da Computação - UFSC

1. INTRODUÇÃO

Nos sistemas de Inteligência Artificial orientados à resolução de problemas é dada muita ênfase ao papel do conhecimento, sendo, por esta razão, comumente chamados de Sistemas Baseados em Conhecimento (SBC)("Knowledge-Based Systems"). De fato, o desempenho e grau de "inteligência" de um SBC estão diretamente ligados à qualidade e usabilidade do conhecimento contido no sistema.Tais sistemas são caracterizados, de modo geral, por manterem uma Base de Conhecimentos explícita onde são armazenadas informações a respeito de um domínio de conhecimento específico [BRACHMAN85]. A ênfase sobre o conhecimento nos SBC fez com que a área de Representação do Conhecimento (RC) assumisse papel fundamental [ALTY 84],[FIKES 85],[WOODS 86],[WAH 89], levando ao desenvolvimento de vários formalismos.

A definição de um formalismo de RC em um SBC determina os tipos de inferências que podem ser utilizados. Tradicionalmente, o formalismo mais usado é o de Regras de Produção (RP). Este formalismo apresenta como vantagens a sua simplicidade sintática e seu apelo intuitivo. Entretanto, RP não provêm facilidades de representação de estruturas mais complexas. Em particular, o seu poder de expressividade é inadequado para descrição de objetos e relações entre eles (tal como objetos compostos) [KOMOSINSKI 90]. Além disso, SBCs que utilizam apenas Regras de Produção como formalismo de representação de conhecimento tendem a apresentar um baixo desempenho quando o número de regras torna-se grande.

Considerando os aspectos acima, percebeu-se que sistemas baseados em um único formalismo de RC (em particular o formalismo de RP) limitam o tipo de informação que pode ser representado e tendem a ficar ineficientes a medida que cresce a quantidade de informações que precisam ser armazenadas. Assim surge uma nova geração de SBC utilizando mais de um formalismo. A combinação típica emprega Regras de Produção e Frames (que se baseia na noção de objeto), formando uma **representação híbrida** que combina as vantagens de ambos formalismos. Os sistemas CENTAUR [AIKINS 85], KEE [FIKES 85], KEOPS [ROCHE 89] são exemplos deste enfoque híbrido.

Com o objetivo de desenvolver SBCs onde a RC e os respectivos mecanismos de inferência sejam definidos pelos formalismos de Regras de Produção e Frames está sendo desenvolvido, no Laboratório de Inteligência Artificial do DCEC, um Ambiente de Desenvolvimento de Sistemas Baseados em Conhecimento (ADSBC). O ADSBC é composto por dois módulos principais: o Módulo de Edição de Base de Conhecimento (MEBC) e o Módulo de Resolução de Problemas (MRP). Através do MEBC, o Engenheiro do Conhecimento (EC) pode editar de forma agradável uma BC (formada por Regras de Produção e Frames) a respeito de um domínio de conhecimento qualquer. Uma vez editada a BC, esta é traduzida de um formato agradável ao EC para um formato que permita a sua eficiente utilização pelo MRP.

O Módulo de Edição de Base de Conhecimento está atualmente em fase de especificação e o Módulo de Resolução de Problemas em fase de implementação (utilizando linguagem PROLOG). Neste artigo são descritos aspectos ligados à RC do ADSBC e ao Módulo de Resolução de Proble-

mas. Comenta-se também as várias possibilidades de combinação dos formalismos de RP e Frames previstas para o ADSBC.

2. REPRESENTAÇÃO DE CONHECIMENTO NO ADSBC

A Representação do Conhecimento (RC) no ADSBC é feita através da combinação de Frames e Regras de Produção. Os itens 2.1 e 2.2 descrevem, respectivamente, a estrutura de um Frame e de uma RP. O item 2.3 descreve as várias possibilidades e vantagens em se combinar os dois formalismos de RC.

2.1. FRAMES

O formalismo de RC de Frames faz com que um determinado domínio de conhecimento seja modelado através da identificação e representação dos objetos que pertencem àquele domínio. O termo objeto deve ser entendido como sendo desde objetos concretos (como por exemplo automóvel, carro, pessoa, etc) até objetos abstratos, situações ou eventos (como por exemplo uma reunião ou um jantar em um restaurante). Em ambos os casos os objetos são caracterizados por um conjunto de atributos ("slots") e, para cada atributo, está associado um valor.

Quando Minsky [MINSKY 75] definiu o formalismo de Frames o fez de maneira muito genérica e sem nenhuma preocupação com uma estrutura que pudesse servir de base para algum sistema [GOMEZ 81]. Isto permitiu que, em cada sistema que utilizasse a idéia de Frames, este fosse definido de forma a atender necessidades específicas. Seguindo esta conduta, no sistema ADSBC, um frame possui uma estrutura própria (descrita em detalhes mais adiante).

No sistema ADSBC, um frame representa uma classe de objetos e uma instância representa um objeto (indivíduo) que pertence a uma determinada classe. Assim, um problema é modelado através da definição de frames e instâncias.

Segundo a semântica definida para o formalismo de Frames no sistema ADSBC, uma instância sempre está consistente com as definições contidas no frame à qual está ligado. Isto significa que todos os slots definidos para o frame estejam preenchidos com valores válidos. Assim, qualquer alteração no valor de um slot só será aceita se este valor estiver consistente com a especificação definida para o slot.

No ADSBC os frames são definidos por fatos PROLOG e apresentam a seguinte estrutura:

```
frame( <nome_frame>,
       <superclasse>,
       <slots_de_documentação>,
       <slots_de_controle>,
       <slots_de_definição> ).
```

A seguir são apresentados em detalhes os componentes da estrutura de um frame.

2.1.1. Nome do Frame

O termo <nome_frame> é um símbolo PROLOG que identifica o nome do frame.

2.1.2. Superclasse

O termo <superclasse> é um símbolo PROLOG que identifica a superclasse do frame. Através da declaração da superclasse é possível construir taxonomias de frames resultando em um poderoso mecanismo de inferência: a Herança. Este mecanismo será explicado no item 3.3.1.

2.1.3. Slots de Documentação

Os Slots de Documentação têm, como o próprio nome indica, a função de documentar o frame onde estão declarados. No ADSBC os slots de documentação definidos são: "author", "date" e "text".
O preenchimento destes slots durante a definição do frame é opcional. Além disso, eles não são herdáveis e não fazem parte do processo de instanciação.

O slot "author" tem como valor um termo PROLOG que reprsenta o nome do autor do frame. O slots "date" tem como valor um termo PROLOG que representa a data de criação do frame. O slot "text" tem como valor um termo PROLOG que descreve a classe de objetos sendo modelada pelo frame.

2.1.4. Slots de Controle

Os Slots de Controle permitem estabelecer condições para que se possa iniciar o processo de criação de uma instância e definir os efeitos colaterais associados à criação (ou não) desta. No ADSBC estão definidos três slots de controle: "to_instantiate", "if_instantiated" e "if_not_instantiated".

O slot "to_instantiate" define condições para que se possa iniciar a instanciação de um frame. O slot "if_instantiated" define ações que devem ser realizadas caso se tenha conseguido instanciar o frame (isto é, todos os slots de definição foram preenchidos com valores válidos). Já o slot "if_not_instantiated" define ações que devem ser realizadas caso não se tenha conseguido instanciar o frame.

A definição dos Slots de Controle é opcional. Quando definido, o valor de um slot de controle é formado por uma expressão (um predicado ou um conjunção de vários predicados) que pode ser avaliada diretamente pelo interpretador PROLOG.

A motivação para a existência dos três Slots de Controle é permitir que o processo de instanciação seja sensível (e também sensibilize) ao contexto onde está ocorrendo, tornando, desta forma, a própria instanciação mais eficiente. Por exemplo, a instanciação de um frame "Analisador_de_Defeito" pode estar condicionada à existência de alguma instância que represente um defeito possível.

2.1.5. Slots de Definição

Os Slots de Definição são definidos pelo Engenheiro de Conhecimento e representam atributos comuns a todos os objetos pertencentes à classe modelada pelo frame.

Como os Slots de Definição exercem o papel principal na representação do conhecimento envolvido na classe de objetos que está sendo modelada, a sua composição difere da composição "slot-valor" dos outros tipos de slots. A idéia é que existam outras estruturas que complementem a descrição "slot-valor". Esta complementação tem os seguintes objetivos :

- definir o valor do slot em tempo de edição (e não em tempo de execução), quando possível;
- restringir o conjunto de valores que poderão ser associados ao slot (em tempo de execução) de forma que o valor seja consistente com a semântica associada ao slot;
- definir como obter o valor de um slot;
- definir quais os efeitos colaterais existentes quando do preenchimento (ou não preenchimento) de um slot.

Para atender os objetivos acima, um slot pode conter as seguintes informações:

- nome do slot (slot_name)
- valor (value)
- valor default (default_value)
- restrição de valor (value_restriction)
- "demon" se_necessário (if_needed)
- "demon" se_preenchido (if_filled)
- "demon" se_nao_preenchido (if_not_filled).

O termo "slot_name" é um símbolo PROLOG e representa o nome do slot, cujo preenchimento é obrigatório.

O termo "value" é qualquer constante PROLOG e representa o valor do slot conhecido em tempo de edição do frame e o slot não pode assumir nenhum outro valor.

O termo "default_value" é qualquer constante PROLOG e define o valor default do slot. Este termo deve ser usado quando o valor do slot é conhecido em tempo de edição do frame e o valor representa uma informação de senso comum, isto é, uma informação que se assume como verdadeira caso não exista nenhuma indicação contrária.

O termo "value_restriction" é uma expressão formada por um predicado (ou conjunto de predicados) que podem ser avaliados diretamente pelo interpretador PROLOG. Um valor é considerado consistente com o slot se a expressão avaliada for verdadeira. O termo "value_restriction" permite restringir os possíveis valores que o slot pode assumir, sendo usado para garantir a consistência do valor do slot.

O termo "if_needed" é uma expressão formada por um predicado (ou conjunto de predicados) que pode ser avaliada diretamente pelo interpretador PROLOG. Caso o resultado da avaliação for verdadeiro, o valor do slot será o valor de uma variável da expressão. Caso isto não ocorra, o slot não será preenchido.

O termo "if_filled" é uma expressão formada por um predicado (ou conjunto de predicados) que pode ser avaliada diretamente pelo interpretador PROLOG. O termo "if_filled" é avaliado caso o slot tenha sido preenchido com valor válido (consistente com as especificações definidas na faceta "value_restriction"). Caso não se tenha conseguido preencher o slot com valor válido, o termo "if_not_filled" é avaliado.

2.2. REGRAS DE PRODUÇÃO

As Regras de Produção (RP) do sistema ADSBC têm a estrutura clássica "SE condições ENTÃO ações". As condições estão normalmente associadas ao estado de uma ou mais instâncias. As ações, por sua vez, podem modificar o estado destas instâncias (alterando o valor de um ou mais slots da instância).

Na abordagem acima, a resolução de problemas é feita de forma tradicional, isto é, identifica-se os objetos pertinentes ao problema e estes são modelados através de frames e instâncias; a seguir modela-se o comportamento destes objetos através de um conjunto de RP. Resolver um problema consiste então em se determinar qual a sequência de ações que transforma as instâncias representando o estado inicial do problema em um conjunto de instâncias que representam o estado final (solução) (encadeamento para frente).

As RP são representadas por fatos PROLOG com a seguinte estrutura:

```
regra ( <nome da regra>,
        <grupo>,
        <condições>,
        <ações> ).
```

O termo <nome da regra> é um símbolo PROLOG e identifica unicamente a regra.

O termo <grupo> é um símbolo PROLOG e identifica que a regra pertence a um determinado grupo de regras relativas a um mesmo problema.

O termo <condições> é uma conjunção de predicados avaliáveis diretamente pelo interpretador PROLOG. Diz-se que as condições de uma regra estão satisfeitas quando todos os predicados que formam as condições são considerados verdadeiros no momento da avaliação.

O termo < ações> é um conjunto de predicados avaliáveis diretamente pelo interpretador PROLOG. Quando uma regra é disparada, cada predicado que faz parte das ações é avaliado.

2.3. COMBINANDO REGRAS E FRAMES

Segundo o enfoque de utilização do formalismo de RP descrito no item 2.2, percebe-se claramente que as regras assumem o papel ativo (principal) em relação ao formalismo de Frames com relação ao processo de inferência (o encadeamento de regras é mais importante que a instanciação). Para caracterizar esta situação utilizamos o termo "Regras Dominantes".

Além do enfoque de utilização de RP descrito acima existe uma outra forma de combinar-se RP e Frames. Nesta outra forma, a qual chamamos "Frames Dominantes", os frames assumem o papel ativo em relação ao formalismo de RP com relação ao processo de inferência (a instanciação é mais importante que o encadeamento de regras). As RP são reunidas em grupos afins e estes grupos são associados à slots. O processo de encadeamento de regras é ativado quando se deseja conhecer o valor de um dos slots da instância que está sendo criada. Desta forma combina-se dois mecanismos de inferência poderosos (instanciação e encadeamento) que permitem unir as vantagens específicas de cada mecanismo num mesmo sistema, tornando-se então um sistema híbrido.

3. MÓDULO DE RESOLUÇÃO DE PROBLEMAS DO ADSBC

O Módulo de Resolução de Problemas (MRP) do sistema ADSBC é composto por um Módulo Interpretador (MI), um Módulo Base de Conhecimento (MBC) e um Módulo Motor de Inferência (MMI) (ver Fig. 1). Estes módulos são descritos a seguir.

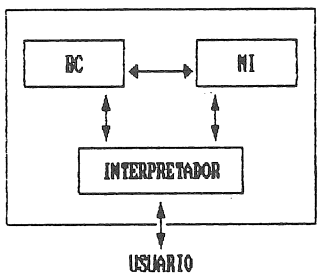


Fig. 1 - Arquitetura do ADSBC

3.1. MÓDULO INTERPRETADOR

O sistema ADSBC funciona dentro do ambiente do interpretador PROLOG em que foi desenvolvido. Assim, para se poder utilizar o sistema é preciso primeiro carregar o interpretador PROLOG. Logo em seguida é carregado o sistema ADSBC que fica armazenado na área de memória gerenciada pelo PROLOG. Este tipo de convivência entre o interpretador PROLOG e o sistema apresenta várias vantagens (descritas a seguir) em relação à facilidade de uso e poder de resolução de problemas.

Na verdade, o sistema ADSBC é formado por um conjunto de predicados cuja sintaxe é idêntica aos predicados PROLOG. Estes predicados permitem a criação de frames, instâncias, consultas à Base de Conhecimento e ativação dos vários mecanismos de inferência (ver exemplo no item 3.3). Desta forma, o Módulo Interpretador tem o papel de traduzir consultas feitas através da sintaxe PROLOG (realizadas pelo usuário do sistema) para uma sintaxe mais específica do sistema ADSBC e encaminhá-las ao Módulo Base de Conhecimento ou ao Módulo Motor de Inferência (ver Fig. 1).

A compatibilidade entre o interpretador PROLOG e o sistema ADSBC apresenta as seguintes vantagens:

- como a sintaxe do ADSBC é igual à sintaxe do PROLOG não existe a necessidade do Engenheiro do Conhecimento/usuário aprender uma nova linguagem. Isto torna a programação mais familiar do que já é conhecido.
- predicados implementados totalmente em PROLOG são naturalmente acessíveis ao sistema ADSBC (e vice-versa). Assim, pode-se explorar as características próprias de cada linguagem.
- o próprio ambiente de programação PROLOG pode ser utilizado para se desenvolver sistemas de resolução de problemas.
- adiciona-se o Método de Resolução como mais um método de inferência disponível para a resolução de problemas.
- pode-se usar a lógica dos predicados como mais um formalismo de representação do conhecimento (além de regras e frames).

3.2. MÓDULO BASE DE CONHECIMENTO (MBC)

Toda informação associada ao domínio específico de conhecimento é armazenada na Base de Conhecimento. A Base de Conhecimento no ADSBC é composta pela Memória de Conhecimento e pela Área de Trabalho, conforme ilustra a Fig. 2.

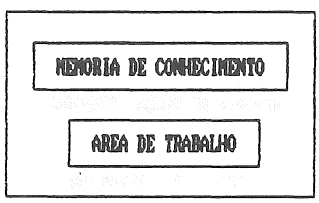


Fig. 2 - Base de Conhecimento

3.2.1. A MEMÓRIA DE CONHECIMENTO (MC)

A Memória de Conhecimento é uma memória onde as informações disponíveis sobre o domínio do problema são armazenadas. Como foi citado no item 2, o conhecimento no ADSBC pode ser representado pelos formalismos de regras e/ou frames. Por esta razão, a Memória de Conhecimento do ADSBC está subdividida em dois módulos, apresentados na Fig. 3, que são a Memória de Regras Permanente e a Memória de Frames. Estes dois módulos representam todas as informações estáticas (permanentes) do problema.

A Memória de Regras Permanente contém o conhecimento sobre o domínio do problema representado através do formalismo de regras (Regras Dominantes).

A Memória de Frames é formada pelos frames (classes de objetos) definidos pelo engenheiro de conhecimento e manipulados na aplicação, e por grupos de regras. Estes grupos de regras são formados pelos

identificadores de regras afins ligadas intimamente com os frames e que têm extrema importância quando se adota o ponto de vista de Frames Dominantes.

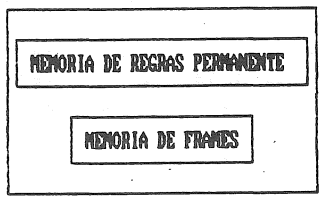


Fig. 3 - Memória de Conhecimento

3.2.2. A ÁREA DE TRABALHO (AT)

A Área de Trabalho é uma memória utilizada para armazenar as informações envolvidas no processo de inferência. No ADSBC a Área de Trabalho é formada pela Memória de Instâncias e pela Memória de Regras Temporária, como ilustra a Fig. 4. Esta divisão é feita pelo mesmo motivo que se divide a Memória de Conhecimento, ou seja, por existir mais de um formalismo de representação de conhecimento.

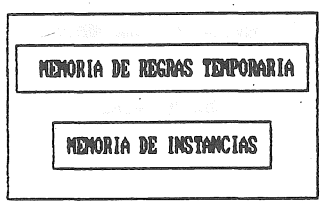


Fig. 4 - Área de Trabalho

A Memória de Instâncias contém todas as instâncias dos frames criados durante a execução da aplicação. A Memória de Instâncias do ADSBC corresponde a Memória de Fatos dos SBC que usam apenas um formalismo de representação de conhecimento.

A Memória de Regras Temporária armazena o conjunto de regras utilizadas no processo de inferência. Esta memória é considerada temporária porque é utilizada no processo de inferência somente quando as informações representadas pelos frames ocuparem um papel de maior importância no contexto (ponto de vista de Frames Dominantes). Nesta situação, as regras armazenadas nesta memória não serão todas aquelas definidas na Memória de Regras Permanente. Somente as regras que fazem parte dos grupos de regras (definidos na Memória de Frames) relacionados com as instâncias estarão presentes na Memória de Regras Temporária.

3.3. MÓDULO MOTOR DE INFERÊNCIA (MMI)

O motor de inferência de um Sistema Especialista (e Sistema Baseado em Conhecimento) constitui-se num conjunto de procedimentos que deduzem informações novas a partir de informações conhecidas [BARR 86]. Portanto, o Módulo Motor de Inferência do sistema ADSBC contém os métodos de inferência inerentes aos formalismos de representação do conhecimento de Regras de Produção e de Frames.

Os métodos de inferência existentes são:

- instanciação e classificação (para formalismo de Frames).
- encadeamento para frente (para formalismo de Regras).

3.3.1. INSTANCIAÇÃO

O processo de instanciação pode ser visto como um casamento (matching) entre um objeto observado (composto por seus slots e respectivos valores) e um modelo de objeto representado pelo frame. O casamento será válido quando todos os slots de definição do frame forem preenchidos com valores consistentes obtidos no objeto observado.

O algoritmo de criação de uma instância do frame F é composto dos seguintes passos:

1. Testar o slot de controle "to-instantiate" (se estiver definido na frame F ou for herdado por alguma superclasse sua).
2. Se o teste do passo 1 for realizado com sucesso então preencher todos os slots de definição definidos no frame F ou herdados por todas as suas superclasses.
3. Se todos os slots de definição foram preenchidos com valores válidos então avaliar o slot de controle "if-instantiated" definido no frame F ou herdado por alguma superclasse de F.
4. Se o teste do slot "to-instantiate" falhou ou algum dos slots de definição do frame F não foi preenchido então avaliar o slot de controle "if-not-instantiated" definido em F ou herdado por alguma superclasse de F.

Como exemplo da aplicabilidade da instanciação pode-se imaginar uma taxonomia de frames, onde cada frame representa um determinado objeto apresentando um tipo de defeito. Assim, instanciar um frame significa verificar se determinado objeto apresenta o defeito modelado no frame.

3.3.2. CLASSIFICAÇÃO

O mecanismo de classificação consiste em criar uma instância para cada frame da taxonomia, partindo do frame raiz até atingir as folhas. A passagem de nível na taxonomia só é feita se o frame do nível superior for instanciado.

Considerando o mesmo exemplo de taxonomia de frames mostrado no item 3.3.1, uma classificação serveria para se determinar quais os defeitos (se existissem) existentes em um determinado equipamento.

Na classificação não se sabe a priori quais frames gerarão instâncias. Este não determinismo é particularmente adequado ao processo

de diagnóstico em geral.

O algoritmo de classificação é semelhante ao algoritmo da instanciação. A diferença está no fato de que, para se realizar uma instanciação de um frame da taxonomia, não é utilizada a herança de slots.

3.3.3. ENCADEAMENTO DE REGRAS

Quanto a utilização das regras de produção, o processo é o tradicional encadeamento de regras. O motor de inferência pode ter dois tipos de estratégia de controle:

- Irrevogável (sem backtracking)
- Revogável (com backtracking).

Em uma estratégia irrevogável, após a geração do conjunto de conflitos, uma regra é selecionada de acordo com algum critério estabelecido e disparada. O processo se repete até que a solução seja encontrada ou nenhuma outra regra possa ser disparada na tentativa de encontrar a solução do problema. Nesta estratégia, o estado em que se encontra a memória de fatos anterior ao disparo da regra é perdido. Isto impossibilita a procura da solução por outro caminho, que não aquele escolhido pelo disparo da regra selecionada, caso a solução não tenha sido alcançada.

Na estratégia revogável existe a possibilidade de se alcançar o objetivo a partir de um outro caminho, pois o estado da memória de fatos anterior ao disparo de cada regra é armazenado.

3.3.4 EXEMPLO

Neste item é mostrado, através de um exemplo simples, como o Módulo de Resolução de Problemas do sistema ADSBC pode ser utilizado. O domínio de conhecimento do exemplo é a Modelagem de Computadores.

Deseja-se modelar um tipo de computador que apresente as seguintes características:

- possua um nome que identifique o computador. Este nome deve ser um símbolo. O valor deve ser perguntado ao usuário.
- possua uma memória cujos valores possíveis são 512, 640, 1024 ou 2048 Kb. Para se conhecer a quantidade de memória de um computador em particular (uma instância da classe computador) deve-se perguntar ao usuário.
- possua um processador cujos tipos possíveis são 80286, 80386 ou 80486. Caso não seja informado pelo usuário o tipo do processador então assume-se o tipo 80386 por default.
- possua um preço cujo valor depende da quantidade de memória e do tipo de processador. O preço é calculado da seguinte forma:

  Se memória < 1024 e processador < 80386
  Então preço = (memória * 2) + processador

  Se memória > 640 e processador > 80286
  Então preço = (memória * 1.5) + (processador * 1.5)

O frame "Computadores", descrito abaixo, modela a classe dos Computadores conforme as especificações anteriores. As informações descritas a seguir estão normalmente armazenadas em um arquivo (por exemplo, "COMPUT.TXT").

```
Frame: Computadores
   author: 'Leandro'
   date  : '10 - 8 -91'
   text  : 'Este frame representa a classe dos Computadores'

   slot nome:
      value_restriction: slot_value(nome,Nome) and
                         symbolp(Nome)
      if_needed: write('Qual o nome ?') and
                 read(Nome) and
                 value_is(Nome)

   slot memoria:
      value_restriction: slot_value(memoria, Mem) and
                         pertence(Mem, [512,640,1024,2048])
      if_needed: write('Quanta memoria ?') and
                 read(Mem) and
                 value_is(Mem)

   slot processador:
      default_value: 80386
      value_restriction: slot_value(processador,Proc) and
                         pertence(Proc, [80286,80386,80486])
      if_not_filled: write('Processador invalido !!')

   slot preco:
      if_needed: rule_groups([grupo_preco]),
                 run_rules,
                 preco(Pr),
                 value_is(Pr),
                 clear_groups  % Limpa MRT %

Rule_Group: grupo_preco

  Rule: pouca_memoria
   IF    slot_value(memoria,Mem) and
         menor(Mem, 1024) and
         slot_value(processador,Proc) and
         menor(Proc, 80386)
   THEN Preco is (Mem * 2) + Proc and
         assert(preco(Preco)) and
         write('Disparada regra pouca_memoria')
```

```
  Rule: muita_memoria
   IF    slot_value(memoria,Mem) and
         maior(Mem,640) and
         slot_value(processador,Proc) and
         maior(Proc,80286)
   THEN Preco is (Mem * 1.5) + (Proc * 1.5)
         assert(preco(Preco))
         write('Disparada regra muita__memoria')

%  o predicado "pertence" (usado nos slots memoria e processador)  de-
senvolvido completamente em PROLOG %

pertence(Elem,[Elem|_]) :- !.
pertence(Elem,[_|Resto]) :-
  pertence(Elem,Resto),!.
```

A seguir é mostrado um trecho de uma sessão de utilização do MRP. O símbolo "?" representa o prompt do interpretador PROLOG. As linhas iniciadas por "%" são comentários explicando o objetivo de cada comando. Assume-se que o interpretador PROLOG, o sistema ADSBC e o arquivo "COMPUT.TXT" já foram carregados.

```
%  para criar uma instância usa-se o comando make. A instância  criada
fica armazenada no fato uma_inst %

? make(computadores,Inst) and assert(uma_inst(Inst))

Qual o nome ? marca1
Quanta memoria ? 1024
Disparada regra muita_memoria

Inst = instance(frame,computadores,
                [nome,marca1,
                 memoria,1024,
                 processador,80386,
                 preco,122115)

% para recuperar o valor de um slot usa-se o comando slot_value %

? uma_inst(I) and slot_value(I,memoria,Mem)
I   = instance(frame,computadores,...)
Mem = 1024

? uma_inst(I) and slot_value(I,nome,N)
I = instance(frame,computadores,...)
N = marca1

% para alterar o valor de um slot usa-se o comando put_value. Quando o
valor proposto é inconsistente com as especificações definidas no fra-
me a consulta falha %
```

```
? uma_inst(I) and put_value(I,processador,68020,Inst)
Processador invalido !! % ativado o demon if_not_filled %
no

? uma_inst(I) and put_value(I,nome ,marca2,Inst)
I = instance(frame,computadores,
             [nome,marca1,
              memoria,1024,
              processador,80386,
              preco,122115)

Inst = instance(frame,computadores,
                [nome,marca2
                 memoria,1024,
                 processador,80386,
                 preco,122115)
```

4. CONCLUSÃO

Neste artigo foi apresentado o Módulo de Resolução de Problemas do sistema ADSBC. Este sistema combina os formalismos de representação do conhecimento de Regras de Produção e de Frames. Por estar contido dentro do ambiente interpretativo da linguagem PROLOG pode-se dispor também do formalismo da lógica dos predicados. A combinação de vários formalismos permite uma representação mais expressiva de cada tipo de informação contido num mesmo domínio de conhecimento.

Apresentou-se também algumas formas de se combinar Frames e Regras de Produção. Percebeu-se que cada frame de uma taxonomia pode representar um contexto de aplicabilidade de um conjunto restrito de Regras de Produção. Assim, combinando-se os mecanismos de inferência de classificação (que envolve o mecanismo de instanciação) com o de encadeamento de regras pode-se obter um desempenho bastante superior à um sistema composto somente por Regras de Produção onde todos competem entre si em igualdade de condições, isto é, em cada ciclo de execução todas as regras devem ser testadas para a geração do conjunto de conflitos.

A perspectiva futura do sistema ADSBC é a sua utilização em problemas de grande porte para que se possa analisar mais precisamente as reais vantagens da utilização de sistemas híbridos.

5. REFERÊNCIAS BIBLIOGRÁFICAS

[AIKINS 85] AIKINS, J.S., "A Representation Scheme Using Both Frames and Rules", Rule-Based Experts Systems, B.G, Buchanan and E.H. Shortlife (eds), Addison-Wesley, 1985.

[ALTY 84] ALTY, J.L. & COOMBS, M.J., "Expert Systems Concepts and Examples", NCC Publications, 1984.

[BARR 86] BARR, A. & FEIGENBAUM, E., "The Handbook of Artificial Intelligence", Vol. 1, Addison-Wesley, 1986.

[BRACHMAN 85] BRACHMAN, R.J. & LEVESQUE, H.J., "Readings in Knowledge Representation", Morgan Kaufmann, 1985.

[FIKES 85] FIKES, R.E. & KEHLER, T., "The Role of Frame-Based Representation in Reasoning", Communications of the ACM, 28(9):904 - 920, septeber 1985.

[GOMEZ 81] GOMEZ, F. & CHANDRASEKARAN, B., "Knowledge Organizations and Distribution for Medical Diagnosis", IEEE Trans. on Systems, Man, and Cybernetics, 11(1):34-42, january 1981.

[HARMON 85] HARMON, P. & KING, D., "Expert Systems - Artificial Inteligence in Business", John Wiley & Sons, INC., 1985.

[KOMOSINSKI 90] KOMOSINSKI, L.J., "Uma Linguagem Centrada em Frames para Desenvolvimento de Sistemas Baseados no Conhecimento", março 1990.

[NILSSON 80] NILSSON, N.J., "Principles of Artificial Inteligence", Tioga Publishing Company, 1980.

[ROCHE 89] ROCHE, C. & LAURENT, J-P., "Les approches objets et le langage LRO2 (KEOPS)", Technique et Science Informatiques, 8(1):21-39, 1989.

[WAH 89] WAH, B.W., et alii, "Computers for Symbolic Processing", Proc. of the IEEE, 77(4):509 - 539, april 1989.

[WATERMAN 86] WATERMAN, D., "A Guide to Expert Systems", Addison-Wesley, 1986.

[WOODS 86] WOODS, W., "Important Issues in Knowledge Representation", Proc. of the IEEE, 74(10):1322 - 1334, october 1986.

coSARA: A Computer-based Environment for Collaborative Design, Analysis and Evaluation of Computer Systems†

Sergio Mujica
Departamento de Ingeniería Informática, Universidad de Santiago de Chile
Casilla 10233, Santiago, Chile

Yadran Eterovic
Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile
Vicuña MacKenna 4860, Santiago, Chile

Steven Berson, Gerald Estrin, Poman Leung, Ivan Tou, Elsie Wu
Computer Science Department, University of California, Los Angeles, CA 90024

ABSTRACT

This paper describes the design and implementation of coSARA a computer-based collaborative environment for the design, analysis and evaluation of computer systems. The general goal of this work is to create a platform that will encourage fundamental and systematic experimentation towards understanding of techniques to realize collaborative systems and methods for collaborative design of computer systems. In this paper we discuss the environment in which coSARA is assummed to operate; we focus our attention on the the object system implementation; we also provide a review of coSARA's user interface and of its tool integration method.

Keywords: collaborative, CSCW, groupware, design methodology, object-oriented, distributed, integrated, user-interface.

1. Introduction

This paper describes the design and implementation of coSARA[11], a computer-based collaborative environment for the design, analysis and evaluation of computer systems. CoSARA has been conceived to be used by a group of engineers working face-to-face, each at a workstation with graphics capabilities, using the methods and tools of SARA[6] to design and realize complex computer systems. coSARA also includes tools and methods to specify and build other computer-based collaborative systems, different from coSARA. *The general, long-term goal of this work is to create a platform that will encourage fundamental and systematic experimentation towards understanding of techniques to realize collaborative systems and methods for collaborative design of computer systems.*

† This work has been partially supported by Hughes Aircraft Co., TRW, NCR Corporation, UNISYS, AT&T, Interactive Development Environments, SUN Microsystems, the State of California MICRO Program, Perceptronics Inc. and FONDECYT Project No. 0437-92.

There are severe technical problems to create useful computer-based systems in support of group work that are not satisfactorily solved today. Some of these problems are concurrent access to data shared by group members, design of effective user interface devices to convey information about group context, methods and tools to compile and examine group work history, and modeling of group processes. A discussion of these problems can be found in [11].

This research has contributed mainly to the technology for the realization of computer-based environments to support collaborative work. Two main contributions are:

- *A distributed object-oriented sdata management ystem* It includes a graphic language for modeling data using concepts of objects and relations, a library of functions for building programs that manipulate objects, support for distributed, collaborative sharing of objects, and persistent storage of objects.

- *A tool modeling and integration method.* This method includes a tool model for environment extendibility and collaborative tool sharing that allows incremental, partial and full integration of tools; enables tools to operate on collaboratively shared objects; and supports the incremental extension of the environment. It also includes a user interface model that allows sharing of interaction mechanisms, provides support for modeling behavioral response to multi-user actions, and enables early testing and analysis of user interfaces.

In this paper we discuss the environment in which coSARA is assummed to operate establishing the area of CSCW [8] that it covers; we focus our attention on the the object system implementation. We also provide a brief review of coSARA's user interface and the tool integration method. We do not present here the graphic language for object oriented modeling of data, details of the user interface model and implementation and details of the tool integration procedure. A full discussion of topics not covered by this paper can be found in [11].

2. Group work

The basic problems of group work are related to communication among group members and coordination of their joint activities. Communication and coordination tasks are required for the group work to progress towards mutually agreed upon goals. These tasks involve complex problems in management of shared information. It is shared information what makes collaboration feasible, expeditious and effective.

Several factors have an impact in the quality of communication and coordination in group work: existence of a common view of the way in which the group works together, a common language for exchange of information among group members, usage of unstructured information (such as natural language, video images and sound), methods and tools for systematic communication and coordination, methods and tools to compile and trace the history of group work.

coSARA approaches these problems offering a group interface which conveys information common to all group members, an object-oriented data model common to all group members and specification of protocols to share objects at the user interface level. These features contribute to create a common language, establish the way in which the group works together and systematize communication and coordination. coSARA lacks methods and tools to manage history of the group process and capabilities to work with unstructured information.

We can classify collaboration according to location and time (Table 1). Asynchronous and remote collaboration are discussed in detail in [11]. In synchronous, face-to-face situations, all group members work in the same room, each one at a workstation, using a common large

screen to focus group attention (images displayed in the large screen may not be same as those displayed in the individual workstations). Group members can communicate by voice and body language. These forms of communication are highly unstructured and difficult to handle by computer. coSARA is designed to be used in synchronous, face-to-face collaboration.

	face-to-face	remote
synchronous	• inmediate response •social interaction • large bandwidth	• inmediate response • computer-based interaction • small bandwidth
asynchronous	•deferred response •computer-based interaction •any bandwidth	

Table 1: Classification of collaborative activities

3. The Operation Environment of coSARA

CoSARA (collaborative System ARchitect Apprentice) was developed at the UCLA Collaborative Design Laboratory, using the hardware and software system shown in Figure 1. CoSARA is written in CommonLisp and it runs under UNIX®. Its size is about 30,000 lines of code. We wrote a few pieces of code in C language to interface TCP/IP system calls. C code amounts to about 100 lines. CoSARA was designed as a classic object-oriented system that includes classes, metaclasses, multiple inheritance, object composition and polymorphic messages [7,3,9]. CoSARA's object system, that we call OREL (Objects and RELations) was built on top of CLOS (Common Lisp Object System) using PCL, a public domain implementation of CLOS created by XEROX PARC [2]. The graphic interface was built using CLUE (CommonLisp User-interface Environment) which is an X Window System toolkit for CommonLisp.

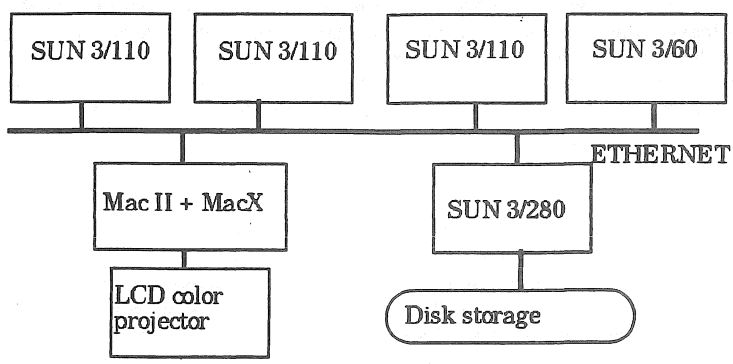


Figure 1: Hardware environment

coSARA is used by a team of engineers for design of complex computer systems in a laboratory shown in Figure 2. There is a large conference table surrounded by workstations. The engineers work looking away from the table to see their workstations. When it becomes necessary that the group discusses a topic, the engineers turn around to face each other at the table. Group

focus on shared information is conveyed by a large screen located at one end of the room. It is, however, desirable that engineers face each other at all times. The massive displays of the workstations prevent such a setup. We expect that technological advances, such as LCD color displays, will help to improve this situation providing high resolution, small and flat screens.

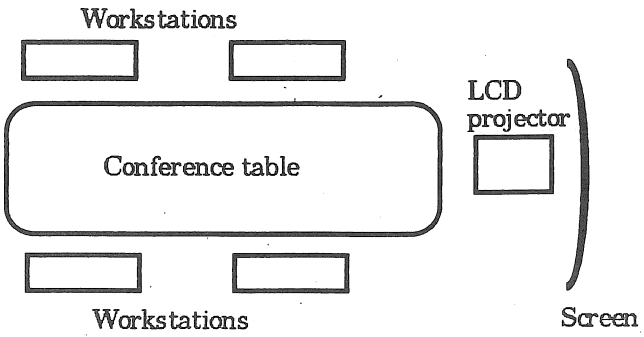


Figure 2: Laboratory layout

4. The design of coSARA

A structural model [6] that shows a modular decomposition of coSARA is shown in Figure 3. The system is first decomposed in two main modules, called ENVIRONMENT and coSARA respectively. ENVIRONMENT encapsulates the assumptions under which coSARA is expected to operate. CoSARA is partitioned into three top-level modules: group-interface, object-world and system-manager.

The behavior of environment is as follows. A user (who is not necessarily a privileged user) initializes the system preparing it for a meeting. Thereafter other group members join the meeting one by one, until a maximum of K (a small integer, usually between 5 and 10) are working jointly. group members operate coSARA using mice, keyboards and graphic displays. A group member can leave a meeting and join it again at a later time. CoSARA creates a new session for each user that joins the meeting, presenting an up to date view of shared objects. All other users are notified of the newcomer. Similarly when a user leaves a meeting all other users are notified. Each group member uses private and shared tools to create modify and examine shared objects in the object-world.

Collaborating group members discuss about shared objects and evaluate modifications made to shared objects, looking at identical images of the objects on their own screens. Figure 4 is an example of how coSARA enables visualization of shared objects. In the example, object-1 is shared by three users who are working at workstation-1, workstation-2 and workstation-3 respectively. Object-1 is drawn in two windows, window-1 and window-2. Window-1 is displayed only on workstation -1, window-2 is displayed on the three workstations. This shows how in user interface elements are also shared objects and therefore several users can share the same window (window-2 is shared by three users in the example). The user at workstation-1 has a private view of object-1 and all three users have a shared view of object-1 on widow-2 .

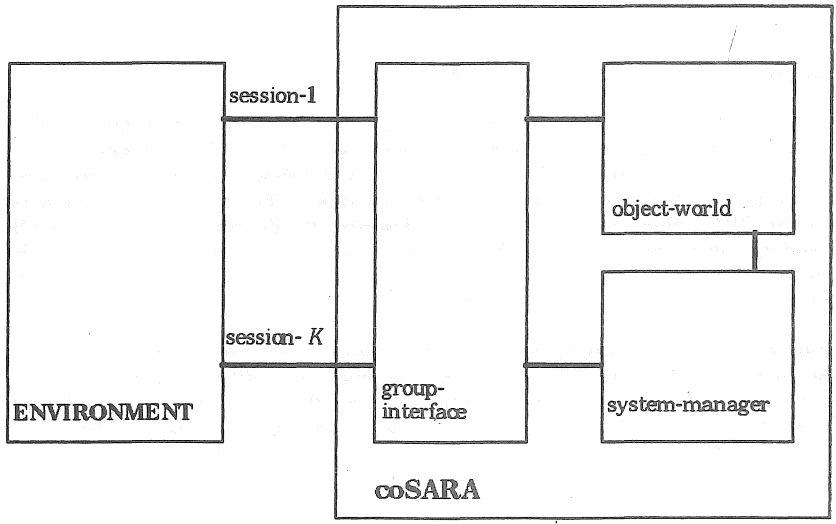


Figure 3: Structural model of coSARA

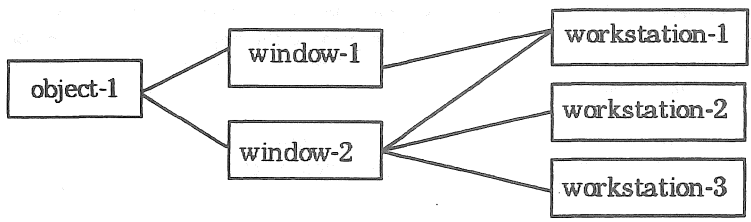


Figura 4: Visualización de objetos compartidos

The group-interface module is a multi-user interface that provides functionality necessary to support group activities. It operates according to protocols for sharing objects at the user interface level. The user interface of a tool specifies a protocol that rules how that tool lets users manipulate objects. The user interface behavior, including collaboration protocols, is specified using SARA models [6].

The object-world module is a realization of the OREL object system and provides functionality for sharing data objects and their operations (note that a tool is an object). Shared objects are stored using a method that we call replication on demand. There are multiple copies of each object, one in each workstation that has read it. This method guarantees that each workstation will store only that subset of objets it is using. Operations done on an object are sent to all workstation that have copies of the object to maintain consistency of data.

The system-manager module allows users to join and leave meetings in progress and the incremental integration of tools into the coSARA environment.

In the following three sections we give a brief description of how coSARA has been designed and implemented. We focus mainly on the object system because we believe it is a fundamental

component of computer-based environments for collaboration. A full discussion of coSARA's design and implementation can be found in [11].

5. Implementation of the object-world module

OREL objects follow a classical model that includes classes, multiple inheritance, polymorphic messages and object composition. Distributed operation of OREL objects takes place in the OREL *object world.* We envision the object world as a storehouse of objects, where users can find and operate on objects in coordination with each other. The object-world module is a realization of the OREL object world. This section describes the distributed operation of OREL objects and its implementation in the object-world module. A full description of the OREL data modeling language and a discussion of design issues concerning coSARA's object system can be found in [11].

Three rules summarize the properties of the OREL object world: the existence of all objects is *knowable* by all collaborating users at all times; an object is *modifiable by only one user at a time,* it can be used otherwise, but not modified by more than one user at a time; changes made to an object that is shared by two or more users, can be seen by other users as they are made, *unless the user who is modifying the object has chosen to restrict such access until changes have been completed.*

Coordination among users is achieved by using long, interactive transactions, controlled by users. To perform some task that will affect shared objects, a user must start a transaction. When the task is finished, or when the user decides that a break is convenient, the transaction is committed and ended. At any point the user may decide to abort the transaction, restoring the objects that were modified to the state they had prior to the beginning of the transaction. A user may have several transactions in existence at any time, but only one of them can be active. The intent is to *provide a way in which the user can organize a set of on-going tasks.*

To execute operations a user acquires hierarchically implied locks on composite objects[1] at any time and in any arbitrary sequence. All locks are released atomically when the transaction is committed to prevent cascading roll-backs, that could be very costly because transactions are long . The locking scheme is a non-strict 2-phase locking protocol[1], that may lead to deadlocks. We rely on the fact that such long, interactive transactions, controlled by human users, allow resolution of deadlocks by social negotiation among users. The object world provides all necessary information to carry on such negotiations.

For example consider the following scenario: designer 1 needs to have exclusive access to modules A and B to perform some task. Designer 1 successfully obtains a lock on module A, but locking module B fails and designer 1 is informed that designer 2 already has a lock on module B. Designer 1 then asks designer 2 for how long she plans to use module B. Designer 2 answers that she will use module B only for a few minutes as soon as she gets a lock on module A. At this time designer 1 realizes that a deadlock exists and informs designer 2 that he already has a lock on A. Designer 1 then agrees to release his lock on A and wait until designer 2 is done. The interaction illustrated by this example is possible because the object world provides information such that designer 1 could identify designer 2 as the current owner of a lock on module B. The interaction was carried out easily because designer 1 and designer 2 were working face-to-face and coSARA provided information about who was using each object.

---

[1] This means that if an object x is composed of objects a and b, then locking x locks a and b as well. The algorithm used to lock composite objects is the simple tree-warning algorithm described by Ullman [1987 (Section 9.7, page 504)].

Work done on any object should be visible to all users at any time, unless whoever is changing the state of the object chooses to hide changes until they are complete. We define a *dirty-read* lock that enables a user to read an object while it is being modified by another user, seeing thereafter the object's changing, intermediate and uncommitted states in real-time. A dirty-read lock also warns the user that the state of the object is unreliable because the user who is making changes may abort his transaction, because delays in the network may not maintain strictly the real-time requirement, or because failures may occur in the communications system. We mean to warn the user who has a dirty-read lock because it is possible that the user is misled by the "dirty" state of objects displayed and proceeds to make decisions based on such an unreliable state, destroying the strictness of the transaction model. Such user mistakes could lead to manual execution of cascading roll-backs. Dirty-reading is meant to be used for discussion purposes and joint decision processes. It is not meant to be used in support of individual decisions. Other types of locks are more conventional: write and read locks for normal operation and write-warning locks in support of the tree-warning algorithm for locking composite objects. Locks are not enforced by the system. The observance of locks is left to the tool implementor. In this way we avoid checking for locks unnecessarily.

There are several possible approaches to build an implementation of the object world, some of which have been used successfully in other systems such as: centralized storage, proxy objects [5], fully replicated objects [13], and finally replication of objects on demand, which is the method that we have chosen for our implementation. In this method, each machine has only a copy of those objects that have been requested locally. All copies of an object which reside in some subset of the machines are maintained up to date.

Broadcast methods are used to operate on shared OREL objects. Let us consider the case when an object is shared by more than one user. We will say that each user has a *replica* of the shared object, and our objective is to keep all replicas identical to each other at all times. The most straightforward way to achieve this sort of sharing objects, is to make the smallest possible change to the object's state be applied to all existing replicas in the system atomically and exclusive of any other operation that may change the object.

There are two problems with the approach just described. Changes may be distracting when they propagate to each user's replica in fine-grained fragments and transactions may require excessively large amounts of CPU cycles and network bandwidth. Each individual change of the state of an object would require at least a 2-phase commit operation at a cost of $3n$ messages, when there are n processes participating in the 2-phase commit and no failures occur (Bernstein, Goodman & Hadzilacos [1987], page 236). This cost is quadratic when there are failures.

To alleviate these problems we have adopted broadcast methods, that operate on all the existing replicas of an object simultaneously but do not include any concurrency control mechanism. Using broadcast methods we give control to the tool implementor to determine the proper grain size of state changes that a collaborating user will see in a shared object. By choosing whether a method is broadcast or not, the tool implementor can control whether the method is executed at a single site or at several sites.

For example, consider the class of shared rectangles. One operation that can be done on rectangles is to reshape them, The user is given two operations for reshaping a rectangle:

try-shape: changes the shape of the rectangle only temporarily and displays the rectangle in its temporary shape. When the another operation on the rectangle is performed, its original shape is used and the temporary one is discarded.

set-shape: takes no argument and replaces the rectangle shape with a temporary shape set by **try-shape.**

Now the tool designer can choose what will be the grain size of changes made to a rectangle that will be seen by all users who have a replica of the rectangle. If both **try-shape** and **set-shape** are made broadcast methods, every shape that a user tries will be seen by others. However, it is possible that the tool designer determines that collaborating users do not need to see intermediate shapes. Then, the tool designer would make **try-shape** a non-broadcast method and **set-shape** a broadcast method. If the process of finding the right shape for a rectangle is intended to be done as a group activity, **try-shape** would have to be a broadcast method.

Sharing objects poses a problem of concurrency control. It is necessary that one user does not destroy other users' work. In the previous example, suppose that **set-shape** is a broadcast method and that **try-shape** is not a broadcast method. If two users working at different workstations issue a **set-shape** at the same time, it is not possible to predict the results. If both executions of **set-shape** are interleaved, it could happen that the resulting shape is composed by the height set by the first user and the width and position set by the second user. Furthermore, one could ask, what is the meaning of two users trying different shapes for the same rectangle? The answer to this question is methodological. It depends on the collaboration process that the tools are intended to support.

In summary, the user must manage transactions, lock objects and negotiate solutions to conflicts using the information provided by the object world. The tool implementor must define the granularity with which remote updating of shared objects is done using broadcast methods and provide functionality to handle conditions raised by locking failures.

The implementation of shared objects, replicated on demand, has a central server and zero or more client sessions. The server provides three management services: locking, persistent object storage and meeting configuration. A client consists of a main Lisp process plus several processes that provide network services needed by the client to send and receive shared objects, and to send and receive requests for operation on shared objects. The sending and receiving of objects between clients is done synchronously. Each time a method is performed on a shared object, a request to apply the method is broadcast to all the other participants using an asynchronous broadcast channel.

The operation of the system requires that a server be running. To join an existing design session, a client tells the server that it is joining and requests from the server a site from which it can get a copy of the current data structures. Clients must join sessions one at a time. This guarantees that each client has an up to date copy of system-wide data structures, therefore the server can choose any arbitrary site to send session data to the newly joined client. If this is the first client to join (and therefore no other site can supply the data structures), the client data structures are initialized to be empty. When a client leaves a session, the server is notified so that it will not refer data structure requests of any newly joining sessions to the departing client. The other clients are also notified so that they can update their tables to prevent sending of requests for objects to the departed client.

We define the *storable* class of objects which is a superclass of all classes of shared objects defined in the object-world. Storable objects[2]can be encoded into an ASCII representation.

---

[2]Here and in the rest of this section, when we use the term storable objects, we will mean any object whose class is a subclass of storable.

This representation is the form in which objects are sent between sites and to the database. Each site can then reconstruct the object from its encoded ASCII representation. There are three levels of storage:

Local cache: contains a set of replicas of objects that are stored in the local workstation. It is implemented using a hash table that provides access to an object using either a unique identifier or a combination of object class and symbolic name, which is not necessarily unique.

Global cache: is the union of all the local caches. Each workstation has a global directory listing all the objects that are stored in some local cache. Each entry of the global directory has a list of the machines that store replicas of the object.

Secondary storage: is resident on disk devices and stores objects permanently. This store is managed by a central server and is accessed through a directory that is maintained by the central server.

There is functionality to read, save, delete, rename and copy objets. For example, the function `read-object` takes as arguments either the unique identifier of an object or its name and type. It will search the local cache, the global cache and the central secondary storage in that order. If the object is found the function returns it otherwise it returns `nil`.

There are two functions to support transmission of objects, through the network, between different machines: `encode-object` which produces an ASCII encoding of an object and `construct-object` that takes as argument a string that contains the encoding of an OREL object and produces an instance of the object.

Data such as numbers, symbols, and strings can be encoded trivially. However, storable objects often contain data that cannot be encoded trivially. Two specific types of data that we have had to deal with are pointers to other storable objects, and site-specific data of objects in the Lisp/X Window System interface. Objects dealing with windows have pointers to several device dependent data structures.

Pointers to storable objects are encoded as a call to `read-object`. For example the following string can be the result of applying `encode-object` to a binary tree, i.e. it is the encoded form of a binary tree. The string `"RA.asa-105558"` is the unique identifier of the root, `"RA.asa-105560"` is the unique identifier of the left subtree and `"RA.asa-105561"` is the unique identifier of the right subtree.

```
"(SEARCH-TREE NIL
      "RA.asa-105558"
      (KEY . 15 )
      (LEFT . (read-object :id "RA.asa-105560"))
      (RIGHT . (read-object :id "RA.asa-105561")))"
```

The string "`RA.asa-105558`" and similar ones are the unique identifiers of the objects. The left and right subtrees have a value which is a function to read in the correct subtrees. The `construct-object` function, when reading in this description and filling in the value for the `left` and `right` slots, will evaluate the calls to `read-object`. This will occur recursively, so if the left and right subtrees have their own subtrees, these will all be read in all the way down to the leaves.

Broadcast methods are implemented in three parts. The first and second parts are run on the local machine, while the third is executed on remote machines only. Broadcast methods are implemented by a Lisp macro called `defbroadcast` that generates three different methods, representing the following three parts: code that implements that method's functionality; an :after method that broadcasts a request to invoke the method over the network; a plain version of the method without an :after method which is the version that other sites will run instead of the original method (if they attempt to run the original method, the system would be caught in an infinite broadcast loop).

We present an example of a class counter object to clarify this concept. Counter objects have one slot which contains their current value. The counter class has several methods defined on it including value, increment, decrement, and reset. Value returns the current counter value. Increment (decrement) adds (subtracts) one to the current counter value. Reset sets the counter value to zero. Increment, decrement and reset all return the counter object. We define the counter class, using CommonLisp and CLOS as follows:

```
(defclass counter (storable)
 ((value :accessor value :initform 0)))
```

In this definition, the class counter is a subclass of storable. It has one slot named value which has an initial value of zero and whose value can be accessed by `value`. We can create an instance of this class with the following instruction:

```
(setf count-1 (make-instance 'counter))
```

We use the `defbroadcast` macro to define broadcast methods. Our definition for the `increment` broadcast method:

```
(defbroadcast increment ((c counter))
     (setf (value c) (+ 1 (value c)))
     c)
```

The broadcast method `increment` takes a counter as an argument, adds one to the counter's value, and returns the counter object. When the `increment` method is invoked, two things happen. First, the actual incrementing of the counter occurs. Then the `:after` method is executed broadcasting this method to other sites. Remote sites execute a version of this method that does not broadcast.

There is a problem that can cause extra messages to be broadcast erroneously. This happens if one broadcast method calls another broadcast method. On receiving the first broadcast, the remote sites will execute the no-broadcast versions of the same method, which calls a broadcast method. That broadcast method has to be smart enough not to broadcast. But the local site must also be smart enough not to broadcast the second method or else all the remote sites will do the second operation twice. This problem is solved having a dynamically scoped variable (named `*sync*`) which is set to true if a broadcast method is currently being executed and prevents further broadcasting.

Another problem is that a single method may apply to several objects. Some of these objects (but not others) may exist on other sites. If a method of this nature is executed, remote sites may apply the (no-broadcast) method to objects that do not exist on that site. There is a `send-args` function that makes sure all objects mentioned in the method's argument list are stored at the sites where the broadcast method will be executed.

6. The tool model and the group interface

CoSARA supports a method for modeling and integration of tools into integrated extensible environments. The user interface model and the tool integration procedure include the case of partial integration of foreign, pre-existing tools. We define an integrated, extensible environment as a system that provides a common user interface and a common data management system for a set of tools, such that a new tool can be added without rebuilding any other part of the system.

When we build a new tool from scratch, we build it as an OREL object and we model its user interface using SARA methods and tools. This process results in a fully integrated tool that makes use of all the capabilities of the comon user interface and the common data management system. The method for modeling and integration of foreign pre-existing tools requires the construction of modules to bridge user interface interaction between the coSARA multi-user interface and the foreign tools as well as the construction of modules to bridge interaction between the foreign tools and the object world. How much of the common user interface and the common data management systems a foreign tool shares with other tools is determined by the corresponding bridges. Given the formalism of OREL we envision providing computer support in the construction of such bridge modules. However, in this current work we have not gone so far as to provide algorithms that would allow such automatic support.

Tools are passive in the sense that they do not take control of the environment as would happen in Unix when a program is started (if you run vi, you talk to vi!) Tools are not programs in this sense. Tools are objects, and provide functionality for the invocation of methods that act on data objects. In our view, tools are artifacts that are plugged to the user interface in order to define specific ways in which the system is going to react to gestures made by designers. A tool provides access to a set of one or more related methods to change or to analyze design models. For example, the GMB Simulator [6] is a tool that provides methods to simulate the behavior of a model, do performance analysis and do interactive debugging. The methods that are accessed by a tool are written in CommonLisp, operate on data defined in the collaborative design environment data model and may invoke other similar methods.

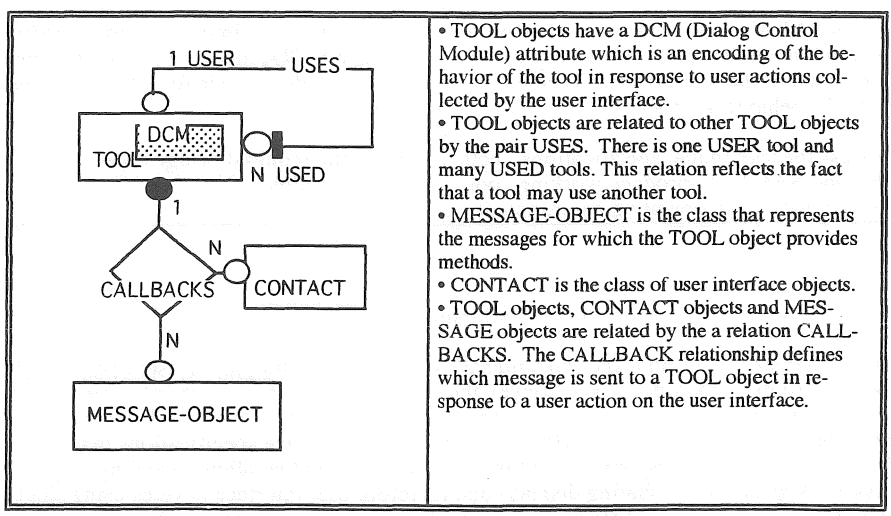


Figure 5 Tool integration procedures

Figure 5 shows a data model of tools, expressed in terms of the OREL graphic language for data modeling. TOOL is the class that encapsulates the properties common to all classes of tools. Specific tools are defined as subclasses of TOOL and therefore inherit all its properties. A particular tool class may have its own data attributes and participate in other relations as we will show in an example later in this section.

To model a particular tool we specialize the class TOOL, defining subclasses of TOOL. Subclasses of TOOL may then participate in relations and have any other arbitrary property. The data model that defines the class of a tool must be integrated into the coSARA data model. This step is done by editing the coSARA data model using the OREL graphic editor.

Table 2 contains the tool integration procedures. The condition that any foreign tool has to meet in order to be partially integrable into the coSARA environment is that the above mentioned bridge modules be constructible.These procedures allow *incremental extension of the environment.* We have shown how the user interface model and construction procedure permits the incremental integration of individual tools into the common user interface. We have shown by way of the procedures stated above that a tool can be incrementally integrated into the coSARA environment provided that we only add to the common data model. If the existing data model is altered in ways other than additions, it may be necessary to rebuild tools and reinitialize the object world database where persistent objects are stored.

Full integration of native tools	Partial integration of foreign tools
1. Modify the common OREL data model as needed by the tool, adding new definitions for the model of the tool object as well as any subsidiary objects that are needed.	1. Modify the common OREL datamodel as needed by the tool, adding new definitions for the model of the tool object as well as any subsidiary objects that are needed.
2. Build a SARA model of the tool's behavior and implement the tool functionality which is included in the interpretation domain of the tool's behavioral model.	2. Specify the user interface bridge modules using a SARA model. Process this specification to obtain code for dialog management.
3. Process the tool model specifications, both OREL and SARA, to produce working code to be included in an operating environment.	3. Build the required bridge object world bridge modules.

Table 2: Tool integration procedures

Severalr esearchers have attacked fundamental issues in supporting multi-user operation. Focus has been on finding good devices to convey the multi-user activity to each individual, how to deal with public and private information, and how to share display devices. However, little or no attention has been paid to formal methods for multi-user interface specifications that allow modeling of behavior in response to multiple user actions and that allow modeling of group processes. Approaches to sharing displays and therefore user interface devices using shared window servers are also unsatisfactory. Use of shared window servers for this purpose results in a relatively easy way to port a single user tool into a collaborative environment but it does not support what we believe is an essential characteristic of collaborative systems: *make each*

individual as aware of other users' actions as possible, as opposed to the aim of conventional systems that intend to give each individual the illusion of being the single user of the system.

Our user interface formalism is based on SARA and OREL. SARA allows us to model concurrent systems and provides tools and methods for early simulation and analysis. By modeling a group of designers as a concurrent system we can incorporate group process models in the user interface. Use of OREL for modeling and construction of user interface objects results in true semantic sharing of user interface objects such as windows, buttons and menus.

The group-interface (Figure 3) is composed of one session per designer working in the environment. The individual sessions that compose the group-interface module do not communicate directly with each other. The different sessions communicate and coordinate by sharing objects that are stored in the object-world module. Our user interface model is based on event processing, operates under external control, has a high bandwidth of communication with the application tools and uses an object oriented programming style.

The process to build and extend the user interface should translate the formal specification of a tool's user interface into working code. Figure 6 displays a dataflow model of the process to build and extend tools that is implemented in coSARA.

Contacts are areas of the screen sensitive to user actions. For example menus, buttons and graphic windows are contacts. Contacts are specified using SARA models. A Contact model can be graphically created and edited using SARA editors, possibly using other predefined contact models stored in the SARA building block library by a separate process. A Tool model can be defined similarly.

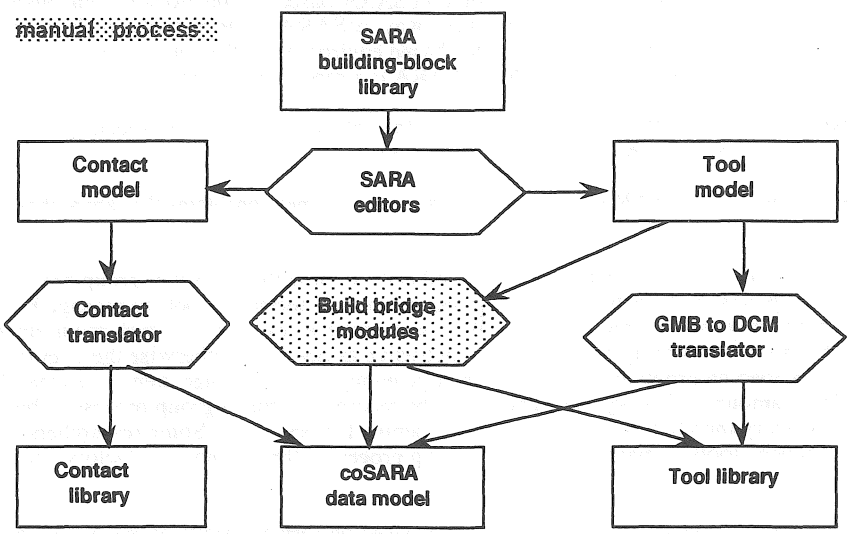


Figure 6: Dataflow model of the process to build and extend tools

The Contact translator processor converts a Contact model into data structures and executable code that implements the contact behavior. It also updates the coSARA data model. The executable code is stored in the Contact library. Similarly, the GMB to DCM translator converts a Tool model into data structures and executable code, to be used by the run time dialog manager, and updates the coSARA data model. The executable code is stored in the Tool library. The Contact library and the Tool library are called collectively *user interface libraries.*

The system is initialized with a set of *basic contacts* (screen buttons, menus, text collectors, dialog boxes, graphers and canvases) and a set of *basic user interface tools,* stored in the different libraries. The contacts implement simple techniques for interaction between the system and the designers, and can be used to compose more complex contacts. The basic user interface tools provide functionality to zoom and pan in a graphic window, select objects pointing to their graphic representation and to display and edit graphic representations of objects.

Basic contacts and basic user interface tools constitute a toolkit for our user interface development system. The dialog manager is provided by SARA in the form of a token machine interpreter, which is part of the SARA GMB simulation tool. Analysis components of SARA (simulation, static control flow analysis and performance analysis tools) provide a set of analysis tools for our user interface development system.

8. Conclusions and future developments

We have identified management of shared objects as a fundamental technology for the realization of computer-based environments that support collaborative work. The design and construction of more powerful and flexible concurrency control mechanisms, will enable us to produce better tools for collaboration, in all possible combinations of time, space and group work style.

As a response to this observation we have built coSARA based on functionality that enables groups to share data and tools in a collaborative way. coSARA contains tools and libraries to specify, analyze and generate other computer-based environments that support collaborative work. From this point of view we say that coSARA is a meta-environment.

The model of the way in which a group uses interface objects, necessary to specify a tool, is a collaboration protocol that represents the group process.

Our research work on coSARA has uncovered a number of new problems that are a challenge for future work:

- The creation of intelligent tools to record and examine the history of group processes, more precisely the acts in which a group commits a decision. Tools to manage history should meet several requirements. We can mention as examples of such requirements that history recording should be as unobtrusive as possible—otherwise the overhead work load on group members may be large enough to discourage them. It is clear that a large amount of history information can be recorded during a group process. It becomes necessary therefore to provide tools to examine history easily. Some researchers have produced tools to manage history of group processes that are not satisfactory yet [4,10,15].

- Creation of concurrency control methods based on formal specifications of collaboration protocols. These methods should allow sharing of data with a granularity that depends on the state of the group process, as modeled by a collaboration protocol. The rules for concurrent access to data that are applied at any time, depend also on the state of the of the group process.

- Creation of a library of building blocks for coSARA, aimed at the modeling of computer-based environments for collaborative work. For example, we need to create building blocks for modeling group interfaces (such as shared dialog boxes) and collaboration protocols.

- It is necessary to study methods to evaluate the impact of introducing computer-based systems in group work.

- We must extend OREL, coSARA's distributed object management system, to operate on wide-area networks since it only works on local area networks.

- Creation of tools for collaborative design of computer systems when groups work remotely and asynchronously.

9. References

1. P. A. Bernstein, V. Hadzilacos & N. Goodman [1987], *Concurrency Control and Recovery in Database Systems*, Addisson-Wesley Publishing Company, Reading Massachusetts.

2. D. G. Bobrow, L. G. DeMichiel, R. P. Gabriel, S. E. Keene, G.Kiczales & D. A. Moon [February 1990], "Common Lisp Object System Specification", Limited Distribution Draft.

3. D.G. Bobrow & M. Stefik [January 1986], "Object-Oriented Programming: Themes and Variations", *The AI Magazine*.

4. J. Conklin & M. L. Begeman [1988], "gIBIS: A Hypertext Tool for Exploratory Policy Discussion", *ACM Transactions on Office Information Systems* 6 (4), 303-331.

5. D. Decouchant [1990], "A Distributed Object Manager for the Smalltalk-80 System", in *Object-Oriented Concepts, Databases, and Applications*, Won Kim & Frederick H. Lochovsky, eds., ACM Press 1990, 487-520.

6. G. Estrin, R. Fenchel, R. Razouk & M. Vernon [February 1986], "SARA (System ARchitects Apprentice): Modeling, Analysis, and Simulation Support for Design of Concurrent Systems", *IEEE Transactions on Software Engineering* SE-12, 293-311.

7. A. Goldberg & D. Robson [1984], *Smalltalk-80: the Language and its Implementation*, Addisson-Wesley, Reading, Massachusetts.

8. I. Greif [1988] *Computer-Supported Cooperative Work:: A Book of Readings*, Morgan Kaufman Publishers, San Mateo, CA.

9. S. Keene [1989], *Object-Oriented Programming in Common Lisp*, Addison-Wesley, Reading, Ma..

10. J. Lee [1990], "SIBYL: A Tool for Managing Group Decision Rationale", Proceedings of the Conference on Computer-Supported Cooperative Work, October 7-10, Los Angeles, California, ACM Press, 79-92.

11. S. Mujica [1991], *A Computer-Based Environment for collaborative design*, Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles.

12. M. Stefik, D. Bobrow, G. Foster, S. Lanning & D. Tatar [April 1987], "WYSIWIS Revised: Early Experiences with Multiuser Interfaces", *ACM Transactions on Office Information Systems* 5, 147-167.

13. M. Stefik, G. Foster, D. Bobrow, K. Kahn, S. Lanning & L. Suchman [January 1987], "Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings", *Communications of the ACM* 30, 32-47.

14. J. D. Ullman [1988], *Database and Knowledge-Base Systems* (Volume I), Computer Science Press, Rockville, Maryland.

15. K. C. Yakemovic & J. Conklin [1990], "Report on a Development Project Use of an Issue-Based Information System", Proceedings of the Conference on Computer-Supported Cooperative Work, 105-118, October 7-10, Los Angeles, ACM Press.

The Sun RPC Language Semantics

Martín Musicante*

Aarhus University
Department of Computer Science
Ny Munkegade Bygn. 540,
DK-8000 Aarhus C, Denmark

UFPE - Departamento de Informática
Av. Prof. Luiz Freire S/N
50.000 Cidade Universitária
Recife - PE - Brazil

ABSTRACT

A formal description of the Sun Remote Procedure Call Protocol is given.
The description is written using the Action Notation style of formal specification.
Action Notation proves to be adequate to express the meaning of the Sun RPC communication mechanism.
KEYWORDS: Formal Semantics, Action Notation, Remote Procedure Call.

Introduction

The Sun Remote Procedure Call Protocol [11], is regarded as a basis for the actual implementation of Remote Procedure Call (RPC) in many distributed computing systems, mainly in those of workstation networks, running UNIX[1].

The RPC concept [1] is a generalization of the procedure call concept, present in most imperative languages. The main idea is that the called procedure is executed in a different environment than the calling program, and possibly in a different machine. Data exchange between the caller and the called processes is performed only by parameter and return value passing. For a comprehensive explanation of the RPC concept, the reader is encouraged to read [1], [6] or [9].

The Sun RPC implementation scheme is based in the existence of a *port-mapper* process in each host. The port-mapper is accessible at a pre-defined, well-know address (the same for all systems) and its purpose is to keep a map of all the "remote accessible programs" available at the local machine.

The port-mapper itself is a remote program, consisting of several remote procedures. Operations such as registration or de-registration of programs, queries for the address of any available procedure in the machine, etc, can be performed using the port-mapper procedures. We refer to a program registered at the port-mapper as an RPC *server*. Each of the remote procedures is called a *service* available at the program. In the same way, a program that calls a remote procedure is said to be a *client*.

The descriptions given in [10] and [11] consist on the definition of a data description language (called XDR, External Data Representation language) and a program definition language (called the RPC Language). The RPC language [11] is an extension of the XDR language.

Each program can contain a set of program versions, and each version can be formed by several procedures (the actual remote procedures).

In this work, Action Notation [3] is used to give a formal semantics of the XDR and RPC languages. Modularity characteristics of Action Notation are used to provide the semantics o

*This work was partly supported by CNPq, Brazil, Grant No. 201.201/7-91. Author's E-mail address: mam@daimi.aau.dk or mam@di.ufpe.br .

[1] UNIX is a trademark of AT&T Bell Laboratories

the RPC language as an extension of the XDR language specification. This is done in a way in which the XDR language semantics can be abstracted from the complete specification, if needed.

Both data representation and program definition languages follow the syntactic specifications given in [10, 11] (with minor changes to correct little errors and to give a better presentation for the semantic functions).

Action Notation

Action Notation [3] is a formal language designed to provide modular and readable descriptions of programming languages. Unlike denotational semantics, where *functions* are used to state the meaning of programs, Action Semantics uses special constructions called *actions*. Like in denotational semantics, semantic functions in Action Notation are defined in a *compositional* way, i.e., the meaning of each phrase of the language depends only on the meaning of its components.

Actions can be performed. They process information in a stepwise way. The performance of an action can yield four possible status: successful completion of execution (*complete*), exceptional termination (*escape*), unsuccessful completion of execution (*fail*) or non-termination (*diverge*).

The description language presented in [3] includes *basic actions* and *action combinators*, that is, rules to obtain new actions from simpler ones. Primitive actions to declare, store and access data and to manage processes and communication are presented also. As it is pointed out in [12], the set of combinators was carefully chosen, in order to facilitate the task of stating the meaning of most programming languages.

Action descriptions are modular. This feature allows the modification of parts of the semantic description without affecting the whole system, as well as the reusability of modules. Action Notation modularity also allows the description to be organized in sections, very much in the same way as chapters and sections in a book.

1 Abstract Syntax

The abstract syntax of both the XDR and RPC languages is described in the following sections. The original description of the languages define some constructions that can be used as abbreviations for common patterns of definition. All these abbreviations were preserved in this description. We choose to consider them in the abstract syntax as a way of maintaining compatibility with the original descriptions.

Action Notation syntactic descriptions can be regarded as defining context-free languages in the same way well-known BNF's does. Usual regular expressions are allowed, as well.

1.1 XDR

The syntax of the XDR language is presented in this section. The description follows (with minor changes) the grammar given in [10].

The symbol '□ ' in the right hand side of the equations corresponding to Constant and Identifier indicates that the non-terminal is left undefined in this part of the specification. Formally, it is an abbreviation for the repetition of the left hand side of the equation.

grammar:

(1) Declaration =
[[Type-specifier Identifier]] | [[Type-specifier Identifier "[" Value "]"]] |
[[Type-specifier Identifier "⟨" Value? "⟩"]] | [["opaque" Identifier "[" Value "]"]] |
[["opaque" Identifier "⟨" Value? "⟩"]] | [["string" Identifier "⟨" Value? "⟩"]] |
[[Type-specifier "*" Identifier]] | "void"

(2) Value = Constant | Identifier

(3) Type-specifier =
[["unsigned"? "int"]] | [["unsigned"? "hyper"]] |
"float" | "double" | "bool" | "void" | [["enum" "{" Enumerand+ "}"]] |
[["struct" "{" Declaration+ "}"]] | [["union" Union-body]] | Identifier

(4) Enumerand = [[Identifier "=" Value]]

(5) Union-body = [["switch" "(" Declaration ")" "{" Union-case+ ("default" ":" Declaration)? "}"]].

(6) Union-case = [["case" Value ":" Declaration]]

(7) Constant-def = [["const" identifier "=" Constant]]

(8) Type-def =
[["typedef" Declaration]] | [["enum" Identifier "{" Enumerand+ "}"]] |
[["struct" Identifier "{" Declaration+ "}"]] | [["union" Identifier Union-body]]

(9) Definition = Type-def | Constant-def

(10) Constant = □

(11) identifier = □

(12) Specification = Definition*

1.2 RPC

The RPC language is presented as an extension of the XDR language. The syntactic description follows (with minor changes) the grammar given in [11].

The semantics of an RPC specification supposes the existence of a procedure body to each procedure declared within a program. It also supposes that this procedure is bound to a name formed by the name of the declared program (but upper-cased), followed by an underscore character, and the version number which the referred procedure belongs to.

The symbol '□ ' in the right hand side of Definition is an abbreviation for the non-terminal itself, meaning that it is being extended at this point.

grammar:

needs: XDR .

(1) Program-def= [["program" Identifier "{" Version-def+ "}" "=" Constant]]

(2) Version-def = [["version" Identifier "{" Procedure-def+ "}" "=" Constant]]

(3) Procedure-def = [[Type-specifier Identifier "(" Type-specifier ")" "=" Constant]]

(4) Definition = □ | Program-def

(5) Specification = Definition*

2 Semantic Functions

Semantic functions are defined to establish the meaning of each phrase of the language. They are stated in a compositional way, by recursion on the syntactic structure of the language.

For space reasons, we present only a tiny but representative part of the semantic descriptions here. The interested reader can refer to the whole work, in [5].

2.1 XDR

This section is devoted to state the XDR language semantics. The definition here is concerned with two main issues: binding syntactic tokens to types and constants, and define translation functions for each type.

2.1.1 Elaborating Specifications

The main purpose of the semantic functions of this section is to create "bindings", i.e: associations from language tokens to semantic entities. The basic binding action is the primitive "bind _ to _", which binds a syntactic token, given as first argument, to a value or cell given as second argument.

The combinator "_ before _" takes two actions. The bindings produced by the first one are merged with the bindings received for the whole action. The result of this merge is passed to the second action (the one after the keyword "before").

The elaboration of a specification consist on the elaboration of the definitions it is composed by. Elaboration is performed by merging the binding maps corresponding to the component definitions.

- elaborate _ :: Specification → action [binding][using current bindings]

(1) elaborate () = complete

(2) elaborate (D_1:Definition D_2:Definition+) = elaborate D_1 before elaborate D_2

2.1.2 Elaborating Definitions

Constant declarations are elaborated simply by binding the supplied token to its corresponding value.

Type declarations are elaborated by binding the token that identifies the type to a pair. The first component of this pair is a "type" semantic value. The second component is an abstraction[2] that, when enacted, performs the translation between byte streams and semantic values.

- elaborate _ :: Definition → action [binding][using current bindings]

The elaboration of a constant declaration consists on the binding of the token corresponding to the identifier I to a semantic value, corresponding to the valuation of the constant.

(1) elaborate ⟦"const" I:Identifier "=" C:Constant⟧ =
bind the token of I to valuation C

(2) elaborate ⟦"union" I:Identifier B:Union-body⟧ =
elaborate B before
| bind the token of I to pair(typified-union of B, closure of abstraction of translate B)

2.1.3 Translating Type Declarations

The main purpose of the XDR data language is to define a standard byte-stream representation for simple and structured data. These sequences of bytes will be passed from one process to another during computation (this method ensures portability and the possibility of communication among programs written in different languages).

[2] An "abstraction" is defined in Action Notation as an item of data encoding an action. An abstraction can be stored, bound, and communicated as normal data. An abstraction can also be performed (or "enacted", following the Action Notation terminology). It corresponds more or less to the notion of procedure present in most programming languages.

In order to use the XDR representation in real applications, translation algorithms need to be provided, for data types of a programming language, to and from byte representation.

A major goal in our semantic description is to obtain XDR and RPC presentations as close as possible to the actually implemented systems, without lacking mathematical rigor. This is the motivation of our choose of a "concrete" byte-stream representation of the semantic "sendable" values, instead of choosing a more abstract way.

Semantic values belonging to simple types, as well as values of structured types are translated into byte streams. In the actual C implementation, supplied with the SunOS operative system, the same routine is used to translate data in both directions. Once more, we accept the price of obscuring our semantic description to keep ourselves close to the actual implementation. Only one translation action for each type is specified. The direction of the translation depends only on the kind of data supplied to the action.

When a value is supplied to the actions defined here, they return a byte stream to represent it, following the informal specification in [10].

When a byte stream is supplied to one of the action defined here, both a value and a byte-stream are returned. The value corresponds to the beginning portion of the supplied byte-stream, interpreted as a value of the corresponding type. The returned byte stream corresponds to the "unconsumed" part of the original stream.

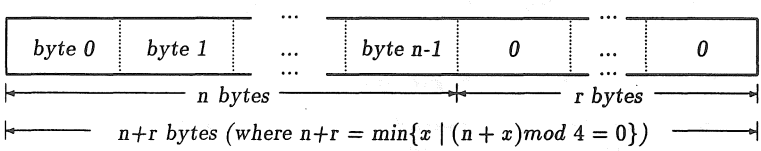
The functionality of the translation semantic functions can be stated as follows.

- translate _ :: Declaration → action [using a given (value | byte-stream)]
  [escaping | giving a (byte-stream | ⟨value, byte-stream⟩)]

The translation of an identified type, is defined as the translation action of it's definition.

(1) translate ⟦*T*:type-specifier Identifier⟧ = translate *T*

Opaque data is data that should be left uninterpreted.
The byte stream representation for fixed-length opaque data of size n, as given in [10], can be depicted as:



As it is specified in [10], the actual length of any byte representation should be a multiple of four (adding null bytes, if needed).
The translation action corresponding to a fixed length opaque data is as:

```
(2) translate ⟦"opaque" Identifier "[" V:Value "]"⟧ =
      | | evaluate V and regive
      then
      | | give the given positive-integer#1
      | | and split the given byte-stream#2 after the given natural#1 bytes
      then
      | | give the given byte-stream#2
      | | and chop the given byte-stream#3 using the given natural#1
      then
      | | give (fixed-length-opaque-value of the given byte-stream#1, the given byte-stream#2)
    or
      | | evaluate V and give component-bytes of the given fixed-length-opaque-value
      | then grow the given byte-stream#2 using the given natural#1
```

The action combinator "_ or _" makes possible the specifiction of non-deterministic actions. The combinator "_ and _" triggers the performance of its component subactions with arbitrary interleaving, while the action "A_1 then A_2" performs first the action A_1, possibly passing some transient data to the action A_2. The primitive action "regive" simply return its received transient data.

When the previous action receives a byte-stream, it is divided in two byte-streams, according to the pre-defined length of the the values of the type (given by the constant V).

The first portion will be encapsulated as an "opaque value". The second portion (possibly null) will be returned (or given) by the action.

When the previous action receives an "opaque value" the corresponding sequence of bytes is extracted and returned (after normalization).

The action "split _ after _ bytes" simply breaks the first argument (should be a byte sequence) in two byte sequences. The number of bytes of the first one is specified as the second argument to the function.

The actions "chop _ using _" and "grow _ using _" perform the operations of cutting and adding the zero bytes necessary to adjust the length of a byte sequence to be multiple of 4, according to a value given in the second argument.

2.2 RPC

The elaboration of a program consist on the binding of its name identifier to its program number, together with the elaboration of all its versions.

Similarly, versions and procedures elaboration bind (version and procedure) identifiers to their corresponding number in the RPC system.

(1) elaborate ⟦"program" *I*:Identifier "{" *V*:Version-def+ "}" "=" *C*:Constant⟧ =
    | evaluate *C*
    then
    | elaborate *V* before bind the token of *I* to the given value

(2) elaborate ⟦"version" *I*:Identifier "{" *P*:Program-def+ "}" "=" *C*:Constant⟧ =
    | evaluate *C*
    then
    | elaborate *P* before bind the token of *I* to the given value

(3) elaborate ⟦T_1:Type-specifier *I*:Identifier "(" T_2:Type-specifier ")" "=" *C*:Constant⟧ =
    evaluate *C* then bind the token of *I* to the given value

2.2.1 Executing Programs

The execution of programs performing the RPC protocol is defined in this section. The same conventions as in the Sun RPC system [11] are adopted; that is: Program, version and procedure names are supposed to be in lower-case (this is ensured by our semantic functions). The existence of an "action" body is supposed for each defined procedure.

The first three semantic equations does not need explanation. The execution of a program *P* consist of two basic parts:

- The initialization of its environment, followed by the elaboration of itself (i.e.: the performance of all its local bindings).
- The instantiation of an Action Notation agent, whose task will be the performance of the program itself (i.e.: the performance of the abstraction that defines the program semantics).

- execute _ :: Specification → action [using current bindings | current buffer] [communicating]

```
(1) execute () = complete

(2) execute T:type-def = elaborate T

(3) execute C:Constant-def = elaborate C

(4) P = ["program" I:Identifier "{" V:Version-def+ "}" "=" C:Constant] ⇒
        execute P =
            | init-environment hence elaborate P
            before
            | | subordinate an agent
            | then send a message [containing closure of abstraction of abstract P] [to the given agent]
```

2.2.2 Abstracting Programs

This semantic function defines the action to be contracted by each agent performing an RPC service.

The main structure of the following action is an iteration. The "unfolding" combinator takes an action (possibly containing "unfold"). The whole action is equivalent to the action given by substituting itself for each occurrence of unfold in the action given as parameter.

The next action performs the first checkings for each received message (RPC call). This checks are for the correct RPC version number and the authentication of the calling process credentials, sending the corresponding error messages to the client process, if any problem is verified. In the normal case, the processing of the message (service required) continues by the performance of the action "perform P". The complete specification of the protocol can be found in [5].

```
• abstract _ :: Program-def → action [using current bindings | current buffer] [communicating]

(1) P = ["program" I:Identifier "{" V:Version-def+ "}" "=" C:Constant] ⇒
        abstract P =
            unfolding
            | | receive a message [from any agent][containing an rpc-msg]
            | then decodify-msg
            | then
            | | | check not (the given natural#1 is 2) and then
            | | | | send rpc-msg(the given xid#9, REPLY, MSG-DENIED, RPC-MISMATCH, 2, 2)
            | | | | to the given agent#8
            | | or
            | | | | check (the given natural#1 is 2)
            | | | and then
            | | | | | give the rest of the given tuple
            | | | | then regive and authenticate-call
            | | | | then
            | | | | | | check (the given boolean#9 is false) and then
            | | | | | | | send rpc-msg(the given xid#8, REPLY, MSG-DENIED,
            | | | | | | | | AUTH-ERROR, the given natural#10)
            | | | | | | | to the given agent#7
            | | | | | | or
            | | | | | | | check (the given boolean#9 is true) and then
            | | | | | | | | | give the rest of the rest of the beginning of the given tuple
            | | | | | | | | then perform P
            | then unfold
```

Conclusion

This article shows that Action Notation can be use to give formal specifications of "real world" applications in a relatively simple way.

We tried to keep the semantic description as close as possible to the actual implementation. This feature imposed a very concrete style of semantic description, in which most of the actual implementation was mirrored. We believe that this relative lack of abstractness (and of mathematic beauty) of our formal specification will facilitate its use as a reference to the actual implementation.

The complete specification given in [5] is about 40 pages of formal description in which each detail of the languages semantics was covered. Our hope is that this work can be viewed as a complementary reference to [10, 11].

Modularity of Action Notation played an important role in the present description. It permitted the specification of the two languages in a very independent fashion, in a way in which the XDR language description can be extracted as an independent entity.

References

[1] A.D.Birrell, B.J.Nelson, *Implementing Remote Procedure Calls*, ACM Transactions on Computer Systems, 2(1):39-59, February 1984.

[2] P.Lee, *Realistic Compiler Generation*, The MIT Press, Foundations of Computing Series, 1989.

[3] P.D.Mosses, *Action Semantics*. Cambridge University Press, Tracts in Theoretical Computer Science Series, 1992.

[4] D.F.Brown, H.P.Moura, D.A.Watt, *Actress: an Action Semantics Directed Compiler Generator*, Technical report, Glasgow University, Department of Computer Science, 1992.

[5] M.Musicante, *The Action Semantics Definition of the Sun XDR and RPC Languages*, Technical Report, Aarhus University, Department of Computer Science (in preparation).

[6] B.J.Nelson, *Remote Procedure Call*, PhD thesis, Carnegie Mellon University, 1981.

[7] J.Palsberg, *An Automatically generated and provably correct compiler for a subset of ADA*. In Proc. ICCL'92, Fourth International Conference on Computer Languages, 1992.

[8] J.Palsberg, *A Provably Correct Compiler Generator*. In Proc. ESOP'92, European Symposium on Programming, 1992.

[9] P.G.Soares, *On Remote Procedure Call*, Area Paper, Department of Computer Science, Columbia University, January 1992.

[10] Sun Microsystems, *XDR: External Data Representation Standard*. RFC 1014, 1987.

[11] Sun Microsystems, *RPC: Remote Procedure Call Specification*. RFC 1050, 1988.

[12] D.A.Watt, *Programming Language Syntax and Semantics*, Prentice Hall International, 1991.

CANGA - UM SISTEMA DE AQUISIÇÃO DE DADOS PARA ANÁLISE DO COMPORTAMENTO ANIMAL EM PASTEJO

Jussara Issa Musse[1], Leandro Fortes Rey[1], Enio Prates[2]

RESUMO

O projeto CANGA surgiu da necessidade de medir o comportamento dos animais ruminantes em pastejo. Seu objetivo é desenvolver e automatizar um sistema eletrônico que colete e analise os movimentos mandibulares dos animais, em tempo real. O sistema é constituido de um transdutor movimento-freqüência, um equipamento eletrônico e um software para recebimento dos dados e emissão de relatórios estatísticos e comportamentais.

PALAVRAS-CHAVE: aquisição de dados, comportamento animal, consumo animal , microprocessadores, transdutores.

1. INTRODUÇÃO

Muito tem sido os esforços a nível mundial na procura de técnicas precisas para a obtenção de estimativas dos parâmetros do valor nutritivo em condições de pastejo. Contudo, até hoje todas as técnicas disponíveis apresentam limitações [1]. O consumo voluntário é considerado a principal limitante do nível e eficiência da produção dos ruminantes sob pastejo. Daí a importância de quantificá-lo nas avaliações das pastagens assim

1 CPD-UFRGS
Ramiro Barcelos, 2574 - CEP 90210, Porto Alegre, RS, Brasil
Fax (55-051) 331 12 15

2 Faculdade de Agronomia - UFRGS
Av. Bento Gonçalves, 7712 - CEP 91500, Porto Alegre, RS, Brasil
Fax (55-051) 331 12 15

como procurar identificar os fatores que o estão determinando em cada condição, de forma a conseguir superar essas limitações e aumentar a produtividade dos sistemas de produção.

A orientação e estabelecimento de corretas práticas de manejo dos bovinos devem ser baseadas no conhecimento do seu comportamento. Dos vários componentes deste comportamento, o conhecimento dos hábitos, em especial, o tempo e os ciclos da atividade diária de pastejo é de primordial importância para recomendação de práticas de manejo visando um maior aproveitamento e eficiência dos animais em pastagens [2] .

Entretanto, a medida do comportamento animal em pastejo pela observação visual é trabalhosa e difícil, senão impossível, pois implica na observação contínua de vários animais ao mesmo tempo [3].

Por isso, foi conduzido um experimento com o objetivo de construir um equipamento eletrônico que registrasse os movimentos mandibulares dos animais em pastejo, e com isso avaliasse o comportamento animal durante as 24 horas do dia. Este sistema de aquisição e análise de dados constitui o projeto CANGA - Coletor de Análise Nutricional de GAdo, que está sendo desenvolvido na Universidade Federal do Rio Grande do Sul, pelo Centro de Processamento de Dados e Departamento de Zootecnia da Faculdade de Agronomia. Este projeto teve origem no sistema descrito em [4][5].

2. DESCRIÇÃO FUNCIONAL

O coletor é um equipamento microprocessado de aquisição de dados, em tempo real, que registra e analisa os movimentos mandibulares dos bovinos em pastejo. Como transdutor é utilizado um oscilador eletrônico com freqüência variável, proporcional aos movimentos mandibulares do animal. O sistema completo (equipamento e transdutor) fica acoplado a um buçal colocado no animal.

O coletor é baseado no microcontrolador INTEL 80C31 com relógio de 3.575611 MHZ, cuja arquitetura básica é mostrada na figura 1.

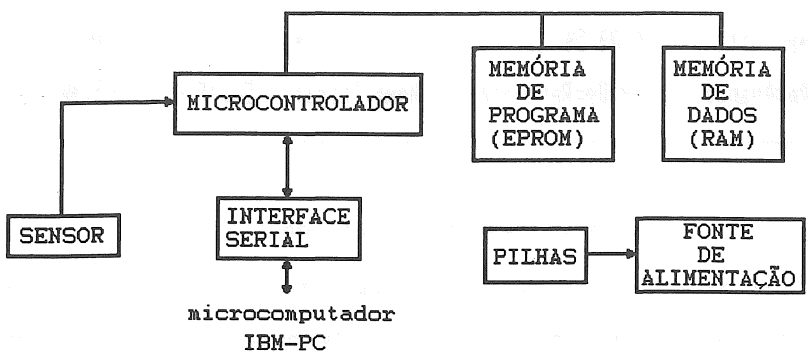


Figura 1. Arquitetura básica do coletor

O equipamento é alimentado com pilhas. A capacidade de coleta, processamento e armazenamento de dados é de até 7 dias contínuos.

Após a coleta de dados, o buçal é retirado do animal e os dados armazenados são lidos pelo SISCANGA, um software que roda

em um microcomputador do tipo IBM-PC, cuja tela principal é apresentada na figura 2.

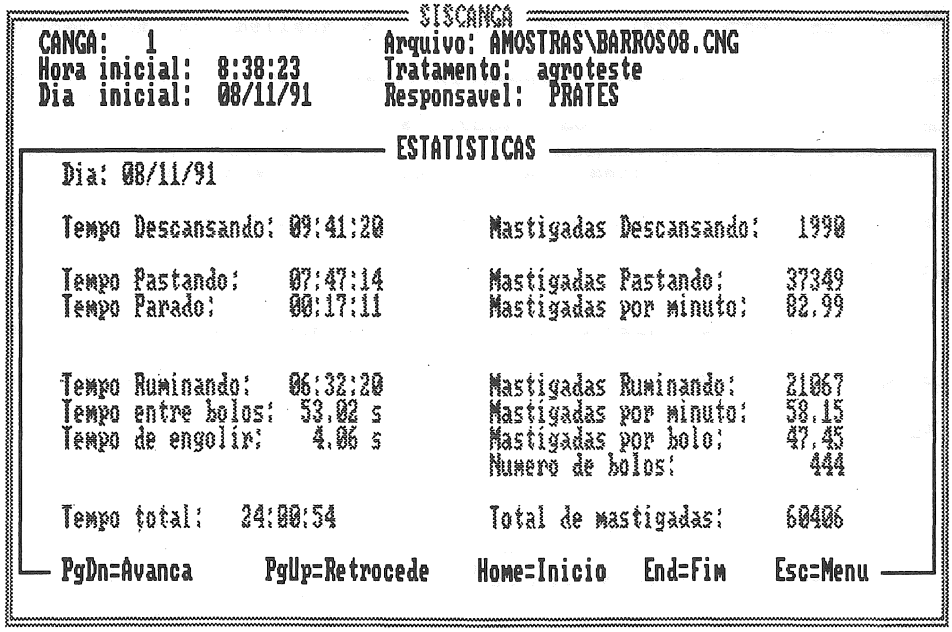


Figura 2. Tela principal do SISCANGA

O SISCANGA fornece relatórios estatísticos e comportamental. No estatístico apresenta, para cada 24 horas, o tempo em que o animal permaneceu pastejando, ruminando ou em descanso, o número de mastigadas por atividade, o número de bolos na ruminação, o número de mastigadas por bolo e o tempo médio entre bolos. Na análise comportamental, o sistema descreve as mudanças de atividade do animal, fornecendo para cada estado as informações supra citadas.

3. METODOLOGIA

O projeto foi desenvolvido em várias etapas, devido a necessidade de obter-se uma maior quantidade de informações sobre a variação do comportamento animal.

Numa primeira etapa foram projetados e construídos o hardware do equipamento e o transdutor de movimento-freqüência. O transdutor é baseado na variação da freqüência de um oscilador LC, onde a indutância é variável. Esta variação é obtida pelo movimento de um núcleo de ferrite de uma bobina de forma a acompanhar os movimentos mandibulares do animal. Com isso, foram obtidas amostras dos movimentos mandibulares das diversas atividades do animal, conforme mostradas nas figuras 3 e 4. Cada ponto destes gráficos corresponde a contagem do número de ciclos do oscilador durante um vigésimo de segundo, que corresponde a uma taxa de 20 amostras por segundo.

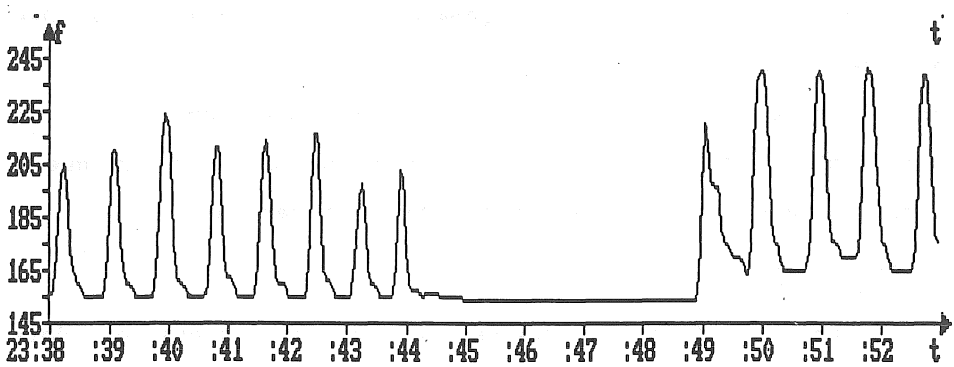


Figura 3. Movimentos mandibulares na ruminação

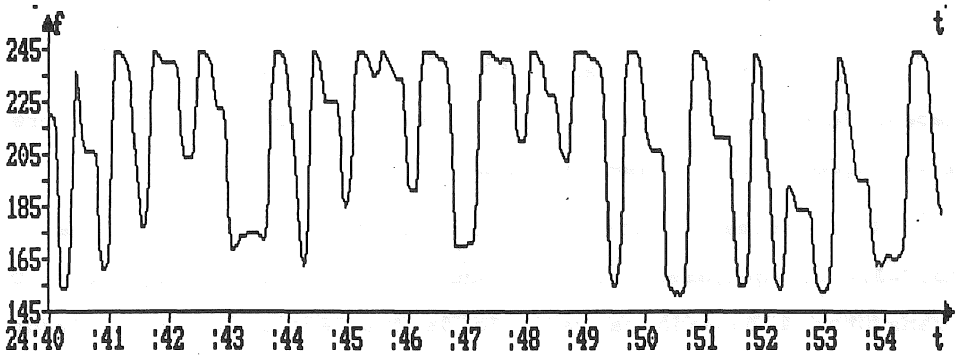


Figura 4. Movimentos mandibulares no pastejo

Após a obtenção de várias amostras, foi possível especificar o software que analisa, em tempo real, os movimentos mandibulares e define a atividade do animal entre repouso, pastejo ou ruminação. O método empregado foi a análise estatística tendo em vista a regularidade dos movimentos na ruminação. Para a validação deste software foram utilizados diferentes classes de animal em distintos dias e condições climáticas. Foram comparadas mais de 1200 horas, com observações visuais e filmagens do comportamento animal, e os resultados permitem concluir que o equipamento fornece estimativas não viciadas do tempo em que o animal permanece pastejando, ruminando ou em descanso.

4. CONCLUSÕES

Durante a fase de testes, o coletor apresentou dois problemas principais, que atualmente estão direcionando a pesquisa.

O primeiro, já esperado, deve-se às condições ambientais adversas em que o equipamento opera: umidade, chuva, variações de

temperatura, vibração, além do risco inerente à imprevisibilidade do comportamento animal. Estes problemas estão sendo gradativamente eliminados com a constante evolução da engenharia de produto.

O segundo, e principal, é o conjunto buçal-transdutor. O ajuste do buçal mostrou-se crítico, exigindo uma boa dose de experiência para sua colocação correta no animal, de forma a não perder movimentos mandibulares. Atualmente, a pesquisa está centrada em um novo transdutor para minimizar o problema de colocação.

A solução final destes problemas viabilizará o repasse desta tecnologia para a indústria criando uma nova ferramenta para os pesquisadores de etologia e nutrição animal.

5. BIBLIOGRAFIA

[1] Y.L.P. LE DU & P. D. PENNING. Animal based techniques for estimating herbage intake. In: Leaver, J.D. ed. Herbage intake handbook. Hurley, British Grassland Society, 37-75. 1982.

[2] R. F. NARDON. Desenvolvimento e Comportamento de Fêmeas de Corte em Pastagens. Porto Alegre, UFRGS, Faculdade de Agronomia, 133p. Dissertação Mestrado Agronomia-Zootecnia. 1985.

[3] P. D. PENNING. A technique to record automatically some aspects of grazing and ruminant behavior in sheep. Grass and Forage Science, Vol 38, 89-96. 1983.

[4] N. C. KRÁS BORGES, D. COREZOLA, E.R. PRATES & F.F. BARBOSA. Desenvolvimeto de equipamento eletrônico para medir o comportamento animal em pastejo. Anais do I Salão de Iniciação Científica, UFRGS, Porto Alegre, RS, 1989, p 30.

[5] N.C. KRÁS BORGES, D. COREZOLA, E. R. PRATES & F.F. BARBOSA. Automatização de sistema eletrônico para medir o comportamento animal em pastejo. Anais do II Salão de Iniciação Científica, UFRGS, Porto Alegre, RS, 1990, p 111.

ANDES: Un Analizador de Desempeño para Programas Paralelos

Oscar Naim y Alejandro Teruel
Departamento de Computación y Tecnología de la Información
Universidad Simón Bolívar
Apartado 89000, Caracas 1080A, Venezuela
e-mail: emsca!usb!teruel@SUN.COM

Resumen: Se presenta una herramienta automatizada que obtiene mediciones para realizar análisis del desempeño de programas paralelos en máquinas MIMD con memoria distribuida. Esta herramienta, denominada *ANDES* (ANalizador de DESempeño), es un *monitor de software* que inserta código adicional para recoger y almacenar información resumida relativa a los *eventos* que definen el desempeño del programa.

La herramienta se apoya en una serie de *métricas* para analizar el desempeño de programas paralelos (aceleramiento, eficiencia, fracción serial empíricamente determinada, porcentaje de tiempo ocioso por procesador, balance de carga, balance de comunicación, tiempo de sincronización y porcentaje de solapamiento de comunicación con cálculo). Se propone una metodología simple de análisis de desempeño que puede aplicarse conjuntamente con ANDES y se presentan brevemente algunos de los resultados de las experiencias vividas con la herramienta.

El prototipo de ANDES fue desarrollado para analizar programas escritos en el lenguaje ANSI C Paralelo de INMOS, sobre un anillo bidireccional de transputadores.

Palabras claves: Paralelismo, desempeño, rendimiento, métricas, C, transputadores.

1. Antecedentes y motivación del trabajo

Los métodos y herramientas convencionales para analizar desempeño en programas secuenciales, no son adecuados en un ambiente distribuido. El esfuerzo que tiene que hacer un programador para realizar el análisis del desempeño de un programa paralelo sin ayuda automatizada es muy grande. El programa a analizar puede contener miles de líneas de código y cientos de rutinas, transformándose en una tarea tediosa y propensa a errores la inserción de nuevo código en el programa, para generar la información relacionada con el desempeño, así como también la de realizar el análisis correspondiente.

Se han reportado herramientas experimentales para programas paralelos con metas similares, tales como [2], [3], [4], [5], [9], [12], [13], [14] y [16], diferenciándose entre ellas por el conjunto de métricas que aplican, el grado de interacción, la calidad de la interfaz visual, tipo de grafos utilizados, nivel de apoyo de hardware y el tipo de máquina paralela a que están orientadas.

2. Descripción y limitaciones de ANDES

ANDES, es un *monitor de software* que analiza el desempeño de un programa insertando código adicional que permite recoger y almacenar información resumida relativa a los eventos que definen el desempeño del programa. Esta información, sin embargo, está dispersa a lo largo de toda la red de procesadores y es local a cada procesador. Una vez que la aplicación termina su ejecución, ANDES recoge y almacena los resultados obtenidos y le ofrece al usuario la posibilidad de analizarlos.

A continuación, se describen las tres partes principales que constituyen ANDES:

a.- *Configurador*

Le permite al usario introducir al sistema los datos particulares de su configuración de hardware, tales como topología y canales de comunicación. Para este trabajo y para los efectos del prototipo, la topología será la de un anillo bidireccional. De esta manera, el configurador se reduce a un archivo de configuración, donde se especifican los canales de entrada y salida de cada uno de los procesadores en el anillo.

b.- *Preprocesador*

Una vez especificados los datos relacionados a la configuración del sistema, el preprocesador se encarga de insertar el código necesario en el programa para recolectar eventos que permitan determinar el comportamiento del desempeño del programa. Estos datos, se resumen en cada procesador hasta que el programa culmine su ejecución, transmitiéndose luego al procesador central. Todas las ocurrencias de eventos en ANDES son *registradas* y *resumidas* en un *árbol dinámico de llamadas* de la aplicación. La rutina principal de la aplicación es la raíz del árbol. Luego, cada vez que se invoca a una *instancia* de una rutina, se crea un nuevo nodo del árbol que enlaza al padre (rutina que invoca) con el hijo (rutina invocada). Cada nodo del árbol, lleva un registro de la información asociada a esa rutina, es decir, el tiempo total de corrida, el tiempo de comunicación, el tiempo de no comunicación, etc. Una vez finalizada la aplicación, este árbol se almacena en disco para luego ser postprocesado.

c.- *Visualizador*

Provee facilidades gráficas para visualizar los resultados obtenidos. La versión del prototipo de ANDES que se presentará en este trabajo no incluye el módulo *Visualizador*. Trabajos futuros extenderán el prototipo para incluirlo.

ANDES posee una serie de limitaciones entre las cuales se encuentran:

- Todos los programas paralelos analizados por ANDES deben ser *finitos*

- ANDES está pensado para analizar aplicaciones que tienen un tiempo de ejecución relativamente grande, es decir, que consumen minutos u horas ejecutándose.

- Como ANDES es un monitor de software introduce cierto grado de *invasividad* en el programa.

- El usuario debe especificar las características particulares de la topología sobre la cual correrá ANDES a través del configurador. En el caso del prototipo, esta topología está restringida a un anillo bidireccional.

- El prototipo de ANDES sólo analiza programas escritos en lenguaje C Paralelo de INMOS [6] y que están contenidos en un sólo archivo.

3. Qué medir y cómo hacer mediciones en sistemas paralelos

Los objetivos principales que se persiguen al realizar mediciones en sistemas paralelos son:

- Conocer el desempeño del programa:
  Le permite al programador conocer si su aplicación está ejecutándose satisfactoriamente o no. De esta manera, el programador determina si vale la pena hacer un análisis detallado del comportamiento de la ejecución de la aplicación, para averiguar las posibles razones por las cuales no se obtiene el rendimiento esperado.

- Hacer proyecciones y determinar escalabilidad:
  Una vez que se conoce el desempeño de la aplicación, es importante plantearse las siguientes preguntas: ¿cómo se comportaría la aplicación si se ejecutara sobre una plataforma con un número diferente de procesadores? Las respuestas a estas preguntas determinan hasta dónde es escalable la aplicación, es decir, hasta qué número de procesadores es factible crecer sin influir negativamente en el desempeño de la misma.

- Ubicar cuellos de botella:
  Si el desempeño de la aplicación no es el esperado, es necesario ubicar aquellas partes del programa dónde se encuentran los problemas que originan la disminución del rendimiento. Una vez que se identifican estos cuellos de botella, se determina dónde concentrar esfuerzos de entonación.

- Obtener información suficiente para conocer e incrementar el grado de paralelismo del programa:
  Ciertas arquitecturas, como los transputadores, permiten realizar comunicación y cálculo de manera simultánea en un mismo procesador, aprovechando al máximo el paralelismo de una aplicación. Sin embargo, ¿se está solapando al máximo el total de tiempo posible en el que se puede hacer comunicación con cálculo? La respuesta puede ser clave en el proceso de entonación de la aplicación.

Con esto presente, describiremos algunas de las *métricas* más útiles: aceleramiento (*speedup*), eficiencia y fracción serial empíricamente determinada [8]. El resto de las métricas que usa ANDES son porcentaje de solapamiento de comunicación con cálculo, porcentaje de tiempo ocioso por procesador, balance de carga, balance de comunicación y tiempo de sincronización y se describen en [10].

El aceleramiento, es el tiempo de corrida del mejor algoritmo secuencial, dividido por el tiempo de corrida de la versión paralela del algoritmo requerido por ρ procesadores.

La eficiencia se define como el cociente entre el aceleramiento y el número de procesadores. Una eficiencia pequeña, indica que se están desperdiciando recursos.

Es obvio que agregando más procesadores se debería reducir el tiempo de corrida, pero ¿en qué medida?. El aceleramiento se encarga de medirlo. Aceleramiento cercano a lineal es bueno, pero ¿qué significa suficientemente cercano a lineal?. Aquí es donde entra en juego la fracción serial. La fracción serial es una medida de la tasa de cambio de la eficiencia y, como tal, permite estimar la proyección del aceleramiento para un número determinado de procesadores.

La fracción serial *teórica* de un programa, es la parte del mismo que debe ejecutarse obligatoriamente de manera secuencial. Calcular de una manera analítica esta fracción serial es, en general, un problema no trivial.

Por esta razón, Karp introdujo en [8] la definición de *fracción serial empíricamente determinada* (fracción serial de aquí en adelante), la cual es una aproximación práctica de la fracción serial teórica. La fracción serial se define de la siguiente manera:

$$f = \frac{\frac{1}{A(n,p)} - \frac{1}{p}}{1 - \frac{1}{p}} \qquad (1)$$

siendo A(n,p) el aceleramiento.

Una ventaja muy importante de la fracción serial, es que permite realizar proyecciones acerca del comportamiento del programa sobre un número mayor de procesadores, determinando la *escalabilidad* del algoritmo.

La fracción serial es una buena herramienta de diagnóstico para detectar posibles problemas en el desempeño de un programa, debido a que mientras el aceleramiento y la eficiencia varían a medida que el número de procesadores crece, la fracción serial podría permanecer constante en un sistema ideal. Variaciones en la fracción serial constituyen valiosas pistas en el análisis de desempeño.

4. Metodología para analizar desempeño

La siguiente metodología tiene como objetivo principal, analizar el desempeño de un programa para determinar si es susceptible a mejoras o no, es decir, aplicarle *performance debugging*. Los pasos de la metodología propuesta son los siguientes:

Paso 1:

Objetivo Responder las siguientes preguntas: ¿qué se tiene?, ¿dónde estamos ubicados?

Métricas usadas: Aceleramiento, eficiencia.

Descripción

El aceleramiento y la eficiencia, permiten determinar si el desempeño es aceptable. Sin embargo, si el aceleramiento no es aceptable, no son suficientes para discernir la causas que originan este comportamiento a pesar que situaciones como las siguientes constituyen *advertencias posibles* de problemas de desempeño:

- El aceleramiento no crece linealmente con el número de procesadores.
- El aceleramiento crece linealmente hasta cierto número de procesadores y, a partir de allí, comienza a disminuir hasta llegar a deteriorar las mejoras de tiempo anteriores.
- El número de puntos de comparación (e.g. 2 y 4 procesadores) es muy pequeño como para poder estimar con precisión si la paralelización del algoritmo ha sido realmente exitosa. Mientras mayor sea el número de puntos de comparación utilizados, nuestras conclusiones estarán más y mejor sustentadas, de lo contrario podríamos llegar incluso a conclusiones erróneas.

Paso 2:

Objetivo Realizar proyecciones y estimar cuánto se puede mejorar.

Métricas usadas: Fracción serial, Ley de Amdahl

Descripción

- Calcular la fracción serial para p=1,..,P (donde P es el número total de procesadores) y obtener su línea de proyección, es decir, cómo se comporta a medida que aumenta el número de procesadores.
- Aplicar la ley de Amdahl, usando la fracción serial apropiadamente, para determinar el máximo aceleramiento posible para un número de procesadores dado.

Es posible que sea difícil determinar si el aceleramiento de una aplicación es lo suficientemente bueno (por ejemplo, un aceleramiento de 70 sobre 100 procesadores ¿es suficiente o no?). Para ello, se utilizan la fracción serial y la ley de Amdahl. A través de estas métricas se puede determinar si, aumentando el número de procesadores, la parte *paralela* del programa crece más rápidamente que la *serial* incrementándose por lo tanto el aceleramiento.

Paso 3:

Objetivo ¿Dónde concentrar esfuerzos? (Análisis por rutina).

Métricas usadas: Balance de carga, balance de comunicación, porcentaje de tiempo de comunicación y no comunicación, porcentaje de solapamiento de comunicación con cálculo, porcentaje de tiempo consumido en sincronizar procesos.

Descripción

- Ubicar las rutinas más *pesadas* de la aplicación, es decir, aquellas rutinas que consuman mayor cantidad de tiempo para saber dónde concentrar esfuerzos.
- Determinar cuellos de botella específicos en esas rutinas.

Paso 4:

Objetivo Aplicar mejoras y repetir 1,2 y 3 para validar los resultados.

Descripción

Una vez que se han identificado los problemas y se le han aplicado los correctivos necesarios a la aplicación, el proceso de evaluación vuelve a comenzar sobre la aplicación ya mejorada. Este proceso de evaluación y mejora iterativo continúa hasta llegar a un resultado satisfactorio con respecto al comportamiento del desempeño de la aplicación.

5. Experiencias con ANDES

Para evaluar ANDES con un problema real y además probar las bondades de la metodología de análisis propuesta, Acosta y Fernández [1] implementaron una versión paralela del algoritmo *Gibbs Sampler* y la analizaron usando ANDES.

El algoritmo del *Gibbs Sampler* tiene como propósito general, obtener una curva que represente la función asociada a una serie de datos iniciales utilizando para ello métodos estadísticos.

Este algoritmo posee un grado de paralelización muy alto, debido en primer la lugar, a la posibilidad de repartir el trabajo a realizar por cada procesador en forma equitativa y, en segundo lugar, a que no existe comunicación entre los procesadores a lo largo de la ejecución del algoritmo y solamente se transmiten los resultados finales.

Los resultados obtenidos por Acosta y Fernández utilizando mediciones directas, fueron los mismos que los obtenidos por ANDES. El aceleramiento, se ubicó entre 3.72 y 3.87 para 4 procesadores (no se disponían de más procesadores para realizar pruebas).

El aceleramiento más bajo, 3.72, fue generado precisamente por uno de los problemas más representativos. Al analizar la fracción serial, esta resultó ser del 2%. Sin embargo, el porcentaje de tiempo ocupado por procesador fue del 96% en promedio y el balance de carga se ubicó en 0.96![0] Estos resultados indican que el balance de carga es muy bueno y pensando que no existe comunicación a lo largo del algoritmo pareciera contradictorio obtener una fracción serial del 2%. Una posible explicación, la cual originó una discusión muy interesante, es que el algoritmo del *Gibbs Sampler* tiene una duración *aleatoria* dependiendo del punto en la secuencia de datos iniciales en que comience. Además, la secuencia

[0] Este resultado se obtiene al dividir el valor promedio entre el máximo, indicando en qué porcentaje nos estamos alejando de la media (el valor ideal es 1).

de instrucciones que ejecuta el algoritmo puede variar dependiendo precisamente de los resultados intermedios obtenidos. Esto quiere decir que el problema que se ejecutó sobre un procesador no es exactamente el mismo que el que se ejecutó sobre 4 procesadores, siendo este último en nuestro caso más *pesado* que el primero, causando esta situación un efecto leve sobre la fracción serial.

En estas y otras pruebas preliminares de ANDES, el overhead causado por las mediciones del prototipo de ANDES llegó a ser del 14% en la ejecución de procesos de más de un minuto). Sin embargo, el overhead para la medición del tiempo de comunicación fue de apenas 2%.

Los resultados de otras experiencias, incluyendo un algoritmo paralelo para la descomposición LU de matrices, son descritos en [10].

6. Conclusiones

- ANDES es un monitor de software que analiza el comportamiento del desempeño de programas paralelos con el objeto de aplicarles *performance debugging*: es decir, mejorar el desempeño del programa a través de un proceso de entonación iterativo. ANDES está basado en las siguientes métricas: aceleramiento, eficiencia, fracción serial, balance de carga, tiempo total de corrida, tiempo de comunicación y no comunicación, tiempo de sincronización entre procesos, balance de comunicación, porcentaje de solapamiento de comunicación con cálculo, porcentaje de tiempo ocioso por procesador
- El prototipo de ANDES es una herramienta útil para realizar análisis de desempeño de programas paralelos, como lo demuestran experiencias como el desarrollo del *Gibbs Sampler* paralelo.
- Con respecto al uso de las métricas propuestas en [10], podemos decir que la incorporación del algoritmo de Jones descrito en [7] para estimar el tiempo en cual un procesador está "ocupado", es decir, realizando labores de cálculo, ha dado resultados muy útiles al ser utilizada junto con la fracción serial empíricamente determinada. También la medición del tiempo de solapamiento de comunicación con cálculo entre dos procesos que se están ejecutando en paralelo en el mismo procesador ha resultado particularmente útil.
- El prototipo de ANDES resultó ser una herramienta muy fácil de usar. En SIMPAR[11], por ejemplo, el proceso de medición se llevó a cabo en tres días, incluyendo el análisis de alrededor de doscientas páginas de números! Sin embargo, las mediciones hechas utilizando ANDES se efectuaron en menos de media hora para el análisis del algoritmo *Gibbs Sampler* descrito en el capítulo V. Aún tomando en cuenta la diferencia de magnitud entre los dos programas (el prototipo de SIMPAR es unas 10 veces más grandes que el *Gibbs Sampler*) la mejora es significativa.
- El overhead causado por las mediciones del prototipo de ANDES llegó a ser del 14%.

7. Recomendaciones y Direcciones Futuras

En base a las experiencias antes descritas hacemos las siguientes recomendaciones:

- Desarrollar el módulo de *visualización*
- Desarrollar un módulo *configurador* que permita extender el prototipo a otras topologías e incluso tratar de hacerlo independiente de la topología.
- Hacer el prototipo más portable e intentar migrarlo a otros ambientes (e.g. transputadores T9000).
- Comparar el prototipo con otras herramientas tanto comerciales como experimentales.
- Determinar el grado de invasividad del prototipo.
- Buscar mecanismos de software, hardware o híbridos más precisos para estimar el tiempo de sincronización entre procesos. Los resultados obtenidos para el prototipo indican que la precisión de la medición es deficiente.

8. REFERENCIAS

[1] D. Acosta, M. Fernández *Generación de Variables Aleatorias en Paralelo* Trabajo de grado de la carrera de Ingeniería de la Computación, Universidad Simón Bolívar. Enero, 1992.

[2] H. Burkhart, R. Millen "Performance-Measurement Tools in a Multiprocessor Environment". *IEEE Transactions on Computers*; Vol. 38, No. 5, May 1989.

[3] A. Couch, D. Krumme *Monitoring Parallel Executions in Real Time* IEEE Computer Society, 1990.

[4] E. Gabber "VMMP: A Practical Tool for the Development of Portable and Efficient Programs for Multiprocessors". *IEEE Transactions on Parallel and Distributed Systems*; Vol. 1, No. 3, July 1990.

[5] W. Harrison "Tools for Multiple-CPU Environments". *IEEE Software*; May 1990.

[6] INMOS. *Ansi C Toolset User Manual* INMOS Limited, 1990.

[7] G. Jones et al. *Measuring the Busyness of a Transputer* Occam User Group Newsletter. No. 12, January 1990.

[8] A. Karp, H. Flatt "Measuring Parallel Processor Performance". *Communications of the ACM*; May 1990, Vol. 33, No. 5.

[9] R. McLaren, W. Rogers *Instrumentation and Performance Monitoring of Distributed Systems* IEEE Computer Society Press, 1990.

[10] O. Naim *Un Analizador de Desempeño para Programas en C Paralelo (ANDES)*. Tesis de Maestría, Universidad Simón Bolívar, Caracas, Venezuela, Enero 1992.

[11] O. Naim, A. Teruel *Consideraciones sobre el Paralelismo en SIMPAR (FASE I)* Reporte IT-1991-001, Depto. de Computación y Tecnología de la Información. Universidad Simón Bolívar, Caracas, Venezuela, 1991.

[12] K. Nichols "Performance Tools". *IEEE Software*; May 1990.

[13] D. Pease, A. Ghafoor, I. Ahmad, D. Andrews, K. Foudil-Bey, T. Karpinski, M. Mikki, M. Zerrouki "PAWS: A Performance Evaluation Tool for Parallel Computing Systems". *Computer*; January 1991.

[14] M. Reilly *A Performance Monitor for Parallel Programs* Academic Press Inc., 1990.

[15] M. Sanabria, A. Hung *Construcción de un Parser de ANSI C* Trabajo de grado de la carrera de Ingeniería de la Computación, Universidad Simón Bolívar, Caracas, Venezuela. Octubre, 1991.

[16] A. Wagner, S. Chanson, N. Goldstein, J. Jiang, H. Larsen, H. Sreekantaswamy *TIPS: Transputer -based Interactive Parallelizing System* Department of Computer Science, University of Columbia, 1991.

CC++ : C++ Concurrente

Alberto Nava

Enzo Chiariotti

Departamento de Computación y

Tecnología de la Información

Universidad Simón Bolívar

Apartado 89000, Caracas 1080A, Venezuela

Abstract

CC++ es una extensión de C++ que provee facilidades para soportar concurrencia. Se incorpora el tipo prototype cuyas instancias, demoninadas *threads*, sólo pueden ser usadas a través de un conjunto de operaciones o *transacciones* . Los threads controlan la forma en la cual las operaciones aplicadas sobre ellos son aceptadas, y es este el único mecanismo de sincronización provisto dentro del lenguaje. También se extienden las nociones de herencia, "freinds" y visibilidad a los prototipos.

Keywords: C++, Concurrencia, Sincronización, Procesos Livianos, Transacciones

1 Introducción

C++ es uno de los lenguajes orientados por objetos más usados en la actualidad, por ser ampliamente disponible y ser compilable en forma eficiente. Desafortunadamente C++ no provee facilidades para programación concurrente.

Se han realizado algunas librerías en C++[3][2] para facilitar este tipo de programación, pero éstas realmente no la soportan en forma completa. También se han desarrollado algunas variantes de C++ que incorporan algunas ideas de concurrencia. cc++[1] incorpora las nociones de "threads" (procesos livianos) ejecutando sobre objetos que equivalen a espacios direccionables distribuidos. Rechazamos cc++ debido a que la granularidad de los objetos no es la más conveniente para algunas aplicaciones, y los threads están soportados en forma completamente diferente que los objetos. C Concurrente[4] (CC) permite compilar programas

LOO[1]	CC++
Objeto	Thread
Clase	Prototipo
Método	Transacción

Figure 1: Esquema Comparativo

en C++, con lo cual se puede utilizar simultáneamente las técnicas orientas por objetos y el soporte a concurrencia por parte del lenguaje. Sin embargo, CC mantiene estos dos mundos separados, lo cual dificulta el desarrollo de muchos programas.

Para añadir concurrencia a C++ nosotros diseñamos un grupo de extensiones que denominamos CC++. Nuestra idea fue extender C++, de forma tal, que un programador entrenado en él pudiese utilizar nuestras extensiones sin mucho esfuerzo.

2 El Lenguaje

En los lenguajes orientados por objetos los objetos son entes pásivos, generados como instancias de clases las cuales se relacionan mediante la herencia. Los objetos son manipulados a través de un conjunto de operaciones definidas en las clases. En CC++ además tenemos los *threads*, como entes activos, que son instancias de *prototipos* los cuales también se relacionan a través de la herencia.

Un thread solo puede ser accedido a través de un conjunto de transacciones definidas en su prototipo. Cuando decimos que un objeto es pasivo nos referimos a que es el llamador el que ejecuta la operación invocada, mientras que en el caso de los threads son éstos los que ejecutan las operaciones sobre ellos invocadas.

[1]Lenguajes Orientados por Objetos

Por ejemplo, para definir un *pipe* síncrono tenemos el siguiente prototipo:

```
prototype Pipe {
public:
    trans void Put(char);
    trans char Take();
private:
    int Inc(int&);
    char *buf;
    int count,size,in,out;
};
```

donde la implementación del *pipe* se basa en un arreglo de tamaño fijo.

Los threads controlan la forma en la cual las operaciones aplicadas sobre ellos son aceptadas, y es éste el único mecanismo de sincronización provisto dentro del lenguaje. Desde el punto de vista del usuario de un thread, una transacción es igual a un función miembro. Pero desde el punto de vista del thread, una transacción es una función miembro sobre la cual se controla su activación. El mecanismo utilizado está inspirado en el rendezvous de ADA, y se extiende con construcciones del tipo *suchthat* y *by* de C Concurrente[4].

El cuerpo del Pipe sería:

```
Pipe::Body()
{
    for(;;)
        select {
            (cout < size) : accept Put();
        or
            (cout > 0)    : accept Take();
        }
}
```

y sus operaciones serían:

```
char Pipe::Take()
{   ++count;return buf[Inc(out)]; }
char Pipe::Put(c)
{   --count; buf[Inc(in)]=c;}
```

Note que las operaciones Put y Take no especifican las sincronizaciones; éstas están en el Body del prototipo.

CC++ es ideal para desarrollar servidores. Por ejemplo para implantar un servidor de nfs[6] tendríamos el prototipo `Nfs_Server`.

```
prototype Nfs_Server {
public:
  lookupres Lookup(lookupargs&);
  readres  Read(fhandle file, fileoffset location, unsiged count);
  writeres Write(fhandle file, fileoffset location, nfsdata data);
private:
}
```

Luego para usar el servidor haríamos `nfs.Lookup(...)`, donde `nfs` es una variable global de tipo `Nfs_Server`.

Note que no hace falta un segundo lenguaje para acceder al servidor. Desde el punto de vista del usuario es equivalente invocar una transacción sobre un thread que esté en el mismo programa o no.

3 Herencia entre los Prototipos

Los prototipos se relacionan a través de la herencia. Esto permite reusar código en forma similar a las clases, pero introduce una nueva dimensión en el reuso. Esta dimensión está representada por la reutilización de especificaciones de sincronización. Es decir, es posible heredar de un prototipo la especificación de cómo se deben ejecutar las transacciones.

Este tipo de herencia junto con la posibilidad de redefinir transacciones en los prototipos es un mecanismo poderoso que facilita los procesos de abstracción y desarrollo.

Por ejemplo para independizar al *pipe* de su representación, tendríamos el prototipo `Pipe` que abstrae el concepto, y prototipos derivados de éste que representan distintas implementaciones.

```
prototype Pipe {
public:
    virtual trans void Put(char);
    virtual trans char Take();
protected:
    Body();
    virtual Boolean Full()=0;
    virtual Boolean Empty()=0;
};

Pipe::Body()
{
    for (;;)
      select {
        (!Full())  : accept Put()
       or
        (!Empty()) : accept Take()
      }
}
```

Este prototipo define su interfaz y las sincronizaciónes necesárias. Las operaciones `Put()`, `Take()`, `Full()` y `Empty()` son virtuales. Es decir, pueden y en éste caso deben, ser redefinadas por los subprototipos.

Note que el `Body` permanece constante para las clases derivadas de `Pipe`.

Ni C Concurrente, ni cc++ permiten herencia entre los prototipos; de hecho cc++ no tiene prototipos ni nada similar. Hasta donde tenemos información no existe un lenguaje basado en C++ que soporte herencia entre prototipos.

4 Transacciones Asíncronas

Inicialmente concebimos las transacciones como síncronas. Es bien conocido que de está forma el software resulta más sencillo, y el la mayoría de los casos se puede emular un comportamiento asíncrono utilizando un generador de ticket y transformando la llamada asíncrona en una notificación de arranque y en una posterior solicitud de confirmación de la completitud de la operación[4].

Sin embargo, es posible que una transacción invoque a otra transacción con lo cual el

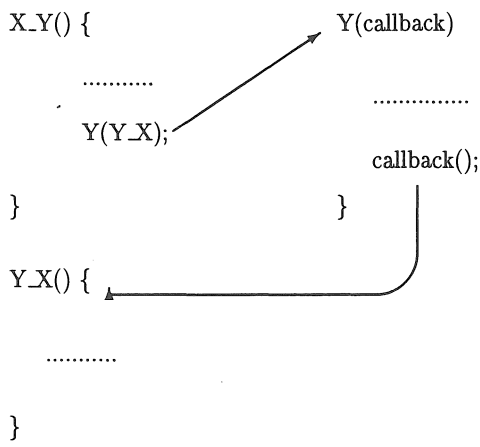


Figure 2:

thread queda impedido de aceptar nuevas transacciones. El problema es similar al de los monitores anidados[5]. Si un servidor invoca una transacción síncrona quedará bloqueado hasta que se complete la transacción, lo cual resulta ineficiente. Por ejemplo el servidor de archivo se bloquearía hasta que se lea un dato del disco, inutilizando por ejemplo el mecanismo de "caching"

Debido a esto incorporamos la noción de llamada a transacción asíncrona, en la cual el llamador no espera por que el llamado acepte y ejecute la transacción. Para especificar que una transacción es asíncrona se le coloca como valor de retorno el token **async**. Por ejemplo

```
prototype Disk_Driver {
public:
   async Request();
}
```

El cuerpo de una transacción X que invoque otra transacción Y queda divide por ésta en dos partes que llamamos X_Y y Y_X. Como muestra la figura 2, la última operación de X_Y es la llamada asíncrona a Y con un nuevo paramétro que es Y_X, de forma tal que al finalizar Y se invoca el forma asíncrona Y_X.

5 Llamadas Anónimas

Otra vez para soportar el desarrollo de servidor introducimos las llamadas a transacciones anónimas, en las cuales no se conoce el nombre del thread llamado. Esto es importante para no ligar sintácticamente los servidor con sus clientes.

Las llamadas anóninas se realizan a través de la noción de apuntador a transacción: Para declarar un apuntador `p` a transación del prototipo `P` que retorna un instancia del tipo `T` hacemos `T (P::*p)()`. Para inicializar `p` hacemos `&a.t()`, donde `a` es una instancia de `P` y `t` es una transacción de `P`. Es decir, `p` representa una instancia y una transacción definida en el prototipo de la instancia en cuestión. Para usar `p` hacemos `(*p)()`. También es posible declarar un apuntador a transacción de cualquier prototipo haciendo T (::*p)(), con lo cual escondemos el tipo de la instancia.

6 Visibilidad de los Miembros de un Prototipo

La visibilidad de la instancias de un prototipo está determinada por la siguiente regla : **una instancia de un prototipo sólo es accesible a través de sus transacciones, esta regla aplica para todos sus clientes. Todo ente distinto de la instancia analizada se considera un cliente de la misma.** Esta regla la llamaremos de integridad.

Un prototipo define un conjunto de transacciones, funciones y objetos. La visibilidad de estas entidades está dada por la aplicación de la regla de integridad a las reglas de visibilidad de C++.

7 Conclusiones

CC++ provee soporte a concurrencia, a través de threads y prototipos. Además permite herencia entre prototipos para facilitar el proceso de desarrollo, y aumentar el reuso. También es posible reusar las especificaciones de sincronización mediante la herencia. CC++ fue implementado sobre el compilador de C++ de GNU lo cual no resultó ser una buena elección, ya que el lenguaje todavía está en una fase de cambios. Actualmente estamos trabajando en un traductor de CC++ a C++.

Referencias

[1] R. Ananthanarayanan. CC++: Preliminary Reference and User Manual. Technical report, Georgia Institute of Technology, Abril 1991.

[2] B. Beck. Shared-Memory Parallel Programming in C++. *IEEE Software*, Julio 1990.

[3] B. Bershad, E. Lazowska, and H. Levy. Presto: A System for Object-Oriented Parallel Programming. Technical report, Computer Science Dept., University of Washington, 1987.

[4] N. Gehani and W. Roome. Concurrent C. *ATT-Bell Laboratories*, Septiembre 1984.

[5] C. Hoare. Monitors: An Operating System Structuring Concept. *Communications of the ACM*, 17(10), Octubre 1974.

[6] R. Sandberg, D. Goldberg, S. Kleiman, S. Walsh, and B. Lyon. Design and Implementation of the Sun Network Filesystem. In *Summer Usenix Conference*, 1985.

Métodos de Aproximación para Problemas de Planificación.

M. Nussbaum, E. Laval, y M. Sepúlveda
Departamento de Ciencia de la Computación
Escuela de Ingeniería, P. Universidad Católica
Casilla 306, Santiago 22, CHILE
FAX (56 2) 552 4054

Keywords: Inteligencia Artificial, Simmulated Annealing, Tabu Search.

Abstract.

Los problemas de planificación generalmente son problemas de optimización del tipo combinatorio. Estos se caracterizan por un espacio de búsqueda donde las posibles soluciones son finitas, o infinitas pero numerables. Aún cuando las posibles soluciones sean finitas, su número puede crecer en forma exponencial con el tamaño del problema. Los métodos tradicionales de optimización, basados en modelos matemáticos, no han podido solucionar un porcentaje importante de los problemas de planificación. En la búsqueda de mejores metodologías han surgido métodos híbridos de la Inteligencia Artificial y la Investigación Operativa los cuales son investigados en el presente trabajo.

1. Introducción.

En la mayoría de las empresas existen problemas de planificación en los cuales se deben tomar decisiones sobre que recursos asignar y como asignarlos. Estos problemas generalmente son problemas de optimización del tipo combinatorio, en los cuales el número de soluciones crece en forma exponencial con el tamaño del problema.
Los enfoques con que se han enfrentado los problemas de planificación tales como los modelos matemáticos, expertos humanos, sistemas expertos [1], no han sido suficientes para resolver este tipo de problemas. Por lo general, las soluciones no son óptimas y tienen tiempos de respuesta elevados.
En la búsqueda de mejores metodologías han surgido métodos híbridos de la Inteligencia Artificial y la Investigación Operativa. En estos métodos se integran heurísticas de búsqueda con algoritmos de optimización.Una línea de investigación que sigue esta tendencia es la de los métodos de Aproximación. Estos se han utilizado con éxito para enfrentar problemas de optimización combinatorios, como se verá en este trabajo.

2. Métodos de Aproximación.

2.1. Descripción.

Los problemas de optimización combinatorios son aquellos que poseen variables discretas y se caracterizan por la búsqueda de un elemento óptimo en un conjunto finito, o infinito pero contable de posibles elementos. Los elementos típicos del espacio de búsqueda son números enteros, conjuntos, permutaciones o grafos [2]. A cada elemento se le asigna un valor real mediante una función de evaluación.
Los Métodos de Aproximación se caracterizan por ir transitando por distintos elementos del espacio de búsqueda, tratando de aproximarse progresivamente al óptimo, para lo cual se escoge uno de los vecinos del elemento actual (generado al realizar un cambio pequeño sobre el elemento actual). De esta definición podemos destacar algunos elementos claves que caracterizarán los distintos métodos de aproximación:
Solución inicial: Los métodos de aproximación comienzan el proceso de aproximación partiendo de un elemento, o solución, inicial.
Vecinos de un elemento: Para transitar de un elemento a otro, se debe definir cuál es la vecindad de un elemento.

Este Trabjo fue finaciado parcialmente por el Fondo Nacional de Ciencia y Tecnología de Chile, proyecto 92842

Elección de la próxima solución: La siguiente solución debe escogerse de entre los vecinos, pero no siempre se deben evaluar todos ellos para escogerla. Es posible focalizarse en un grupo de vecinos mediante algún criterio de muestreo estadístico, o alguna heurística de selección.
Criterio de detención: Los métodos de aproximación continuarán transitando de una solución a otra hasta que se cumpla un cierto criterio de detención.

2.2. Métodos existentes.

En la literatura existen diversos métodos, ya tradicionales, que entran en la categoría de métodos de aproximación. Un método muy simple, es el método de Hill Climbing el cual se caracteriza por escoger al mejor de los vecinos mientras se mejore la evaluación con respecto al elemento actual. Este método tiene la gran desventaja de quedar atrapado en óptimos locales [3]. Entre los diversos métodos de aproximación algunos tienen la cualidad de evitar los óptimos locales, entre elos cabe destacar Simulated Annealing y Tabu Search.

3.3.1. Simulated Annealing.

El método de Simulated Annealing [4] asimila el proceso de búsqueda de una solución óptima con el fenómeno físico de annealing. Este fenómeno se refiere al proceso en el cual se busca el estado de mínima energía de un sólido, disminuyendo lentamente su temperatura desde un estado líquido hasta que se encuentre cristalizado [5].
En el campo de problemas combinatorios el método es el siguiente:

- Se tiene un espacio de búsqueda con posibles soluciones del problema, una de las cuales es la óptima.
- Se escoge una solución inicial y se mantiene un parámetro (la temperatura en un valor alto)
- Se escoge un vecino de la solución actual en forma aleatoria. Este vecino será aceptado como una nueva solución de acuerdo con el siguiente criterio:
  - Si el vecino tiene una mejor evaluación que la solución actual (se acerca al óptimo), se acepta con una probabilidad 1.
  - Si el vecino tiene una peor evaluación que la solución actual (se aleja del óptimo), será aceptado con una probabilidad que depende de la temperatura y de cuánto empeore la evaluación.

El valor del parámetro que representa la temperatura es inicialmente alto, de manera que se acepten vecinos malos con una probabilidad cercana a 1. A medida que evoluciona el proceso la temperatura disminuye lentamente hasta llegar a 0, caso en el que no se aceptan vecinos malos. De esta manera se realiza un recorrido por distintas zonas del espacio de búsqueda, sin quedar atrapado en óptimos locales.
A continuación se describe la manera en que Simulated Annealing responde a las interrogantes generales de los métodos de aproximación que se plantearon previamente:
¿ Cómo escoger la solución inicial ? Debido a la alta aleatoridad inicial del proceso, no tiene sentido escoger una "buena" solución inicial, o sea una solución que sea cercana al óptimo. Basta con una solución inicial aleatoria.
¿ Cómo se definen los vecinos de la solución actual ? El método no tiene asociada una definición de vecindad, está dependerá de cada problema en particular.
¿ Cómo escoger la próxima solución de entre esos vecinos ? Se escoge en forma aleatoria un vecino y es aceptado de acuerdo a la función probabilística descrita anteriormente.
¿ Cuándo detenerse ? La condición de termino depende del problema en particular, usualmente se refiere a un límite de iteraciones efectuadas o a iteraciones sin obtener mejoras en la solución.

3.3.2. Tabu Search.

El método de Tabú Search [6][7] también se basa en acercarse al óptimo mediante aproximaciones sucesivas. Una de las principales características de este método es el utilizar un mecanismo explícito para evitar quedar atrapado en óptimos locales: la lista tabu. Estas son soluciones que no pueden escogerse durante un cierto número de iteraciones. En cada iteración se genera el conjunto de vecinos de la solución actual, de este conjunto se eliminan aquellos que pertenezcan a la lista tabu, y se escoge el "mejor" de ellos. La nueva solución se acepta, aún cuando esta tenga una peor evaluación que la solución antigua. Cada nueva solución escogida se incorpora a la lista tabú, evitando así retroceder por caminos recientemente recorridos. En el contexto del Tabú Search, la lista tabú es conocida como un procedimiento de memoria de corto plazo. Otros métodos de memoria, de más largo plazo, también son utilizados para buscar la intensificación y diversificación de la búsqueda.

En base al método básico de Tabu Search se responden a continuación las interrogantes sobre los métodos de aproximación:
¿ Cómo escoger la solución inicial ? En Tabu Search una "buena" solución inicial permitirá obtener resultados en forma más rápida. Generalmente se utilizan métodos heurísticos relacionados con el problema para obtener soluciones más cercanas al óptimo que las determinadas aleatoriamente.
¿ Cómo se definen los vecinos de la solución actual ? Dependerá de cada problema en particular y de la manera en que se modelen las soluciones.
¿ Cómo escoger la próxima solución de entre esos vecinos ? Antes de escoger el vecino óptimo se descartan los vecinos pertenecientes a la lista tabu.
¿ Cuándo detenerse ? La condición de termino depende del problema en particular.

3. Problemas de planificación resuelta con un enfoque de aproximación.

3.1. Descripción.

Dada la complejidad de los problemas de planificación y los resultados poco satisfactorios obtenidos con las técnicas clásicas, se han buscado nuevos enfoques. El que se presenta a continuación corresponde a utilizar Métodos de Aproximación para resolver problemas de planificación, y cumple con los siguientes requisitos básicos:

- Un modelo para representar problemas de planificación en forma estándar.
- Un conjunto de estrategias concretas para realizar la aproximación.
- Un grado de flexibilidad para adaptar el sistema a cada problema específico.

3.2. Modelo Genérico: Secuencia de elementos.

Se desea utilizar una misma herramienta para distintos problemas de planificación. Para ello se deben representar los distintos problemas de una manera estándar, utilizando un modelo genérico para una gran cantidad de problemas de planificación. El modelo genérico que se utilizó fue el de una secuencia de elementos, a la cual se le busca una permutación óptima. Este modelo se aplica a distintos problemas de planificación ya sea en forma directa o indirecta utilizando un mapeo del problema a uno de secuencias.

3.3. Estrategias concretas de búsqueda.

Las estrategias de búsqueda corresponden a la manera en que se definen los elementos claves de los métodos de aproximación (las preguntas definidas anteriormente). El sistema planificador debería proveer alternativas concretas para responder estas preguntas, de tal manera que el usuario no se preocupe de cómo implementar la búsqueda para cada problema en particular. En los siguientes puntos se describen las estrategias de búsqueda que se han definido para el sistema planificador, sin perjuicio de que en el futuro se vayan desarrollando nuevas estrategias.

3.3.1. Generación de la secuencia inicial.

La secuencia inicial puede ser de gran importancia para el desempeño de la búsqueda. En general una secuencia inicial con una evaluación cercana al óptimo, permitirá acercarse con mayor rapidez al óptimo. Los siguientes son los criterios para definir la secuencia inicial:
Secuencia Aleatoria: Ordenar los elementos de la secuencia al azar.
Utilizar Secuenciamiento Parcial: Esta es una heurística genérica basada en la heurística NEH [referencia] que consiste en comenzar con una secuencia de un solo elemento e ir insertando nuevos elementos que se ubican en la posición que otorgue la mejor secuencia parcial.
Secuencia sugerida por el usuario: El usuario define la heurística.

3.3.2. Definición de vecinos.

Los vecinos de una secuencia son todas aquellas secuencias que se geñeran al aplicar un cambio pequeño sobre ella. Los vecinos son definidos de acuerdo a tres tipos de cambios básicos que se aplicna a la secuencia actual:
Cambios tipo Swap: Intercambiar los elementos que están en dos posiciones dadas.
Ejemplo: (A B C D E F) swap(5,2) -> (A E C D B F)
Cambio tipo Shift: Insertar un elemento de una posición en otra posición.
Ejemplo: (A B C D E F) shift(5,2) -> (A E B C D F)
Cambio tipo Invert:Invertir la subsecuencia comprendida entre dos posiciones
Ejemplo:(A B C D E F) invert(5,2) -> (A E D C B F)

3.3.3. Como escoger el siguiente vecino.

La manera en que se escoge el siguiente vecino es uno de los puntos más determinantes en el proceso de búsqueda. Los criterios serán divididos en tres tipos:

Reducción del espacio de búsqueda local. Se restringe el número de vecinos en casos en los cuales el número de vecinos es elevado o la evaluación de cada uno es lenta. Para reducir el espacio de búsqueda local se propone hacer muestreos aleatorios.
Uso de inteligencia Se refiere a métodos que permitan orientar la búsqueda, sugiriendo algunos vecinos que deberían, o no, escogerse. En particular están los métodos explícitos para evitar óptimos locales. El primero de éstos son las las listas tabú, el cua consiste en almacenar una lista que identifique las secuencias visitadas en las últimas t iteraciones, las cuales no podrán ser visitadas. Un segundo método es prohibir malos, en el cual se supone que si se realiza un cambio en una secuencia y con éste se generan otra secuencia con muy mala evaluación, este cambio es prohibido por un cierto número de iteraciones.
Criterios de aceptación. Se refiere al cómo escoger la siguiente secuencia una vez que ya se ha reducido el espacio de búsqueda y se han aplicado criterios de inteligencia. Se proponen tres criterios. Escoger al mejor vecino: Se escoge al mejor vecino independiente de sí este mejora o empeora la evaluación actual. Utilizar Hill Climbing: Se escoge al mejor de los vecinos siempre y cuando éste mejore la evalución de la secuencia actual, en caso contrario se mantiene la secuencia actual. Aceptación Probabilística: Este es el criterio utilizados en Simulated Annealing. El mejor de los vecinos se aceptará si mejora la evaluación con respecto a la secuencia actual. En caso contrario se aceptará con cierta probabilidad.

3.3.4. Cuándo detenerse.

Se proveen los siguientes criterios de detención. Límite de tiempo: Fijar la cantidad máxima de segundos para que el sistema realice aproximaciones. Límite de iteraciones: Fijar la cantidad máxima de iteraciones a realizarse. Resultado alcanzado: Detenerse al llegar a una solución óptima, o cercana al óptimo. Resultado alcanzable: Realizar una estimación de cuanto más será posible aproximarse al óptimo. Esta estimación puede realizarse de acuerdo a la velocidad de aproximación. Estos criterios se utilizan en conjunto, produciéndose la detención al satisfacerse el primero de ellos.

3.4. Flexibilidad.

El sistema de planificación propuesto provee de una serie de estrategias y parámetros distintos para realizar la planificación. Si se utilizará exactamente la misma estrategia de búsqueda para todos los problemas, se tendría un sistema extremadamente rígido que puede entregar resultados muy buenos en algunos casos, pero en otros casos otorgar resultados malos o en forma ineficiente.
Se propone que el usuario pueda escoger estrategias para adaptar el sistema a sus problemas concretos. La elección de estrategias puede hacerse de acuerdo a la experiencia acumulada por otras personas, o bien 'sintonizando' el sistema mediante sucesivas prueba. En la práctica el usuario dispone de un Script donde especifica especifica los criterios utilizados para escoger la solución inicial, escoger los vecinos de una solución, escoger la próxima solución de entre los vecinos y el criterio de detención.

4. Implementación de la Shell.

Con el fin de implementar la metodología propuesta en la sección anterior y ofrecer al usuario una herramienta que le permita enfrentar cualquier tipo de problema en forma expedita, se desarollo una shell.

4.1. Descripción de la Shell.

4.1.1. Componentes.

En la siguiente figura se aprecia un esquema del sistema:

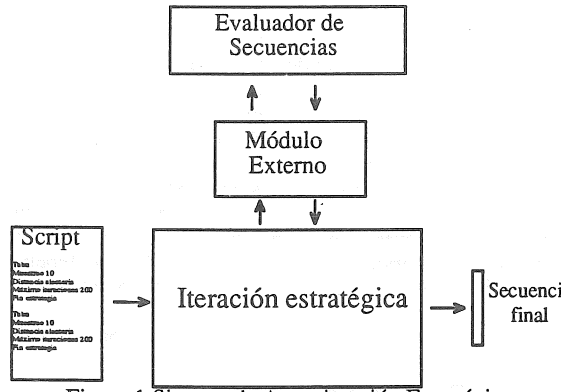


Figura 1 Sistema de Aproximación Estratégica

Iteración estratégica. Componente principal formada por una serie de módulos que permiten realizar cada iteración de acuerdo a las estrategias definidas por el usuario.
Script. Archivo de texto en el cual el usuario puede especificar las estrategias que desea utilizar para realizar la búsqueda. Para ello se provee de un lenguaje declarativo que le permite disponer de los distintos criterios como más le convenga. La Figura 2 ilustra un ejemplo.

```
tipo_movida swap
muestreo 5
distancia_aleatoria
tabu 7
prohibir_malos 4
máximo_iteraciones 1000
máximo_sin_mejora 50
máximo_rango 10
fin_estrategia
```

Figura 2. Script que especifica las estrategias a utilizar
Módulo Externo. Permite la comunicación del módulo principal con el planificador de secuencias (sistema externo independiente).

4.1.2.Funcionamiento.

Con el fin de utilizar la shell, el usuario debe realizar los siguientes pasos:
Modelación del problema. Modelar el problema de manera tal que la planificación dependa de la existencia de un secuenciamiento de elementos. El evaluador de secuencias debe ser capaz de evaluar la bondad de cada secuenciación.
Creación del módulo externo. Crear las funciones componentes del módulo externo descrito en la sección anterior. Una vez realizada esta interfaz, el usuario dispondrá rápidamente de una herramienta para solucionar su problema.
Script. Una vez establecida la comunicación del evaluador con el sistema, el usuario puede establecer cual estrategia es la que permite abordar mejor su problema utilizando el script.
Output. Los resultados de la planificación son desplegados en una serie de archivos de salida que contienen la secuenciación óptima del problema el valor óptimo encontrado.

4.1.3.Ambiente de desarrollo.

La shell fue desarollada en lenguaje C, en ambiente Unix. La plataforma de hardware utilizada fue una SUN SPARC Station 1.

4.2. Ejemplo Real: Asignación de tareas a máquinas.

Descripción.
Una empresa debe realizar N trabajos cumpliendo ciertas restricciones de fechas de inicio y término.Para realizar los trabajos se dispone de M máquinas. Cada trabajo se descompone en una serie de etapas, cada una de las cuales debe realizarse en una máquina por un período determinado de tiempo. Se tiene la restricción de que las máquinas no pueden utilizarse para más de una tarea

simultáneamente, y éstas requieren un tiempo de puesta en marcha entre tarea y tarea. Distintos trabajos pueden requerir las mismas máquinas. De esta manera no todos los trabajos se podrán realizar en cualquier instante, sino que dependen de la utilización de las máquinas por el resto de los trabajos. El problema consiste en asignarle a cada máquina un calendario de tareas de manera que se puedan cumplir todos los trabajos con el mínimo tiempo de retraso. Un enfoque para este problema es el planificar sólo una tarea a la vez, de acuerdo con la disponibilidad de maquinas. Esta asignación se puede hacer con una heurística simple. Luego el problema se transforma en determinar el orden en el cual se deben hacer las asignaciones de las N tareas, osea encontrar una secuencia óptima de N elementos. En este caso la evaluación de cada secuencia implica todo un proceso de planificación de cada elemento en forma individual. La función utilizada para evaluar la planificación es la Tardanza Cuadrática Acumulativa (TCA), la cual depende de los días de atraso de cada trabajo. Esta función castiga los días de atraso en forma polinomial.

Modelación.

De acuerdo a lo señalado en la sección 4.1.2, los elementos son los trabajos pedidos para los cuales se debe encontrar un secuenciamiento óptimo.

Determinación de estrategias más apropiadas.

Generación de la secuencia inicial. Para la generación de la secuencia inicial se utilizó el secuenciamiento parcial. Los resultados obtenidos permitieron generar un buen punto de partida, en algunos casos muy cercanos al óptimo.

Tipos de cambio. Para seleccionar el tipo de cambio se realizaron pruebas utilizando distintas movidas. El Gráfico 1 muestra los resultados obtenidos los cuales sugieren que la estrategia adecuuada al problema es preferir movidas del tipo swap.

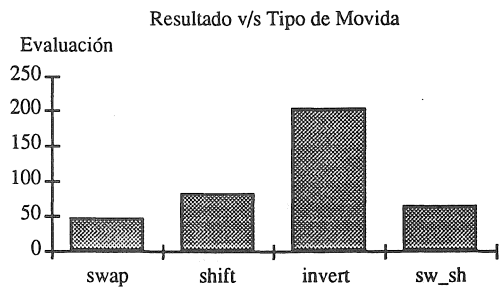


Gráfico 1. Tipos de movida

Reducción del espacio de búsqueda local. Se analizaron ocho tipos de muestreo (A...H) para determinar la mejor forma de acotar el espacio de búsqueda. Estos se constrastaron con el valor óptimo posible de la función de costos (0) y el tiempo de CPU requerido. (Gráfico 2).

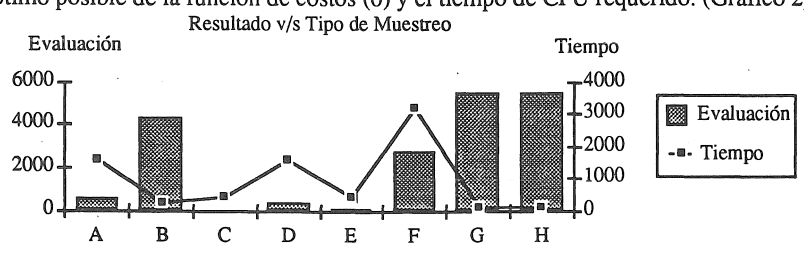


Gráfico 2. Tipos de Muestreo

Los resultados obtenidos sugieren que para este problema se debe utilizar un muestreo del tipo C, el cual consiste en escoger 10 elementos de la secuencia y evaluar la factibilidad de hacer un swap de ellos con otras 5 posiciones de la secuencia.

Uso de inteligencia. Gráfico 3 ilustra los resultados de aplicar listas tabú y listas de prohibir malos. Se realizaron pruebas con listas tabú de distintos tamaños comprobándose que para el problema en estudio es preferible no utilizar dichas listas. Distinto es el caso con listas de prohibir malos donde, para este problema, es conveniente fijar el período de prohibición en 3 iteraciones.

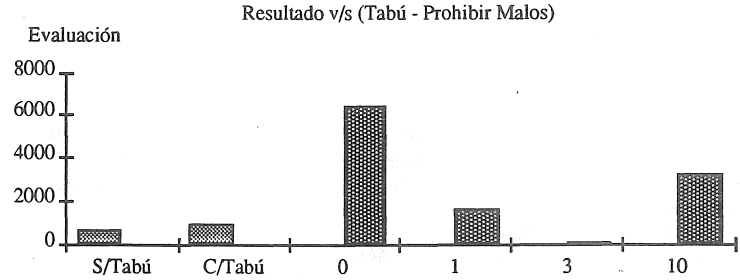


Gráfico3. Uso de inteligencia

Criterios de Aceptación. Es necesario decidir si se avanza o no a la mejor secuencia encontrada entre las vecinas. Las pruebas realizadas (Gráfico 5.4) determinan que es conveniente, para el problema estudiado, aceptar el nuevo estado en forma probabilística.

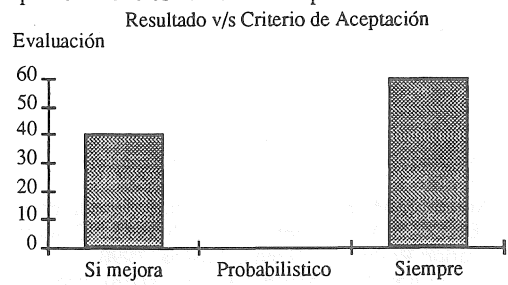


Gráfico 5.4. Criterios de Aceptación

Cuándo detenerse. Haciendo un estudio de todas las pruebas, en las cuales se utilizaron límites relativamente grandes, se llega a los siguientes valores:

Límite de tiempo	1800 seg
Límite de iteraciones	3000
Límite de iteraciones Sin Mejora	600
Resultado óptimo	0

5. Conclusiones

En este trabajo se presenta un enfoque muy simple y poderoso a la vez para enfrentar problemas de planificación. El sistema propuesto permite enfrentar en forma sencilla diversos problemas de planificación obteniendo excelentes resultados. Estos son obtenidos con bajo costo de desarrollo al no tener que desarrollar un planificador a la medida. Los resultados obtenidos con los problemas reales han validado la utilidad del sistema y los métodos de búsqueda que están bajo él.

En el futuro cercano se espera incorporar mayor inteligencia a los métodos de búsqueda a fin de hacer más eficientes los procesos. En el sistema actual el usuario debe tener un gran conocimiento tanto del problema como de la herramienta optimizadora. Un desafío es que el sistema se sintonice automáticamente para operar con la mejor estrategia para diferentes problemas de planificación. De esta manera, el usuario podrá efectivamente desentenderse del cómo se resuelven los problemas.

6. Bibliografía.

[1]: Parra E. and Nussbaum M., "A Production Scheduling System", Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile, Casilla 6177, Santiago, Chile, 1990. Enviado al Journal on Computing Operations Research Society.

[2]: Papadimitrou C., Steiglitz K., "Combinatorial Optimization: Algorithms and Complexity", Prentice-Hall, Englewood Cliffs, N.J.,1982.

[3]: Glover F., "Candidate List Strategies and Tabu Search", 1989a.
[4]: Kirkpatrick S, Gelatt Jr. and Vecchi M.P., "Optimization by Simulated Annealing", Science 220, 671-680, 1983.
[5]: Laarhoven P.J.M van, "Theorical and computational aspects of simulated annealing", Ph. D. thesis, Erasmus Universiteit Rotterdam, 1988.
[6]: Glover F. "Tabu Search-Part I", Orsa Journal on Computing, Vol 1, No 3, 190-206, 1989b.
[7]: Glover F. "Tabu Search-Part II", Orsa Journal on Computing, Vol 2, No 1, 4-32, 1990.
[8]: de Werra D. and Hertz A., "Tabu Search Techniques: A Tutorial and Application to Neuronal Networks", OR Spektrum 11, 131-141, 1989.
[9]: Dueck G. and Scheuer T., "Threshold Accepting. A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing", 1988.
[10]: Faigle U. and Kern W. , "Some Convergence Results for Probabilistic Tabu Search", Memorandum No 822, Faculty of Applied Mathematics, University of Twente, 1989.
[11]: Glover F. and Greenberg H., "New Approaches for Heuristic Search: A Bilateral Linkage with Artificial Intelligence", European Journal of Operational Research, vol. 39, No. 2, 119-130, 1989.
[12]: Glover F. and Laguna M., "Target Analysis to Improve a Tabu Search Method for Machine Scheduling", Technical Report, Advanced Knowledge Research Group, US West Advanced Technologies, Boulder, Colorado, 1989.
[13]: Goldberg D., "Genetic algorithms in search, optimization, and machine learning, Addison-Wesley Pub. Co., Inc, 1989.
[14]: Hertz A. and de Werra D., "The Tabu Search Metaheuristic: How we used it", Research Report ORWP 88/13, Ecole Polytechnique Fédérale de Lausanne, Departement de Mathématiques, 1988, to appear in Annals of Mathematics and Artificial Intelligence.
[15]: Khuri S. and Batarekh A., "Genetic Algorithms and Discrete Optimization", International Conference On Operation Research, Vienna, Austria, 1990.
[16]: Kuik R. and Salomon M., "Multi-level lot-sizing problem: Evaluation of a simulated annealing heuristic", European Journal of Operational Research 45, 25-37, 1990.
[17]: Laguna M., Barnes J. and Glover F., "Scheduling Jobs with Linear Delay Penalties and Sequence Dependent Setup Costs and Times Using Tabu Search", Research Report, Department of Mechanical Engineering, The University of Texas-Austin, 1989.
[18]: Nawaz M., Enscore E.E. and Ham I., "A Heuristic Algorithm for the m-machine, n-Job Flow Shop Sequencing Problem", OMEGA, Int. J. of Mgmt Sci, vol 11, No 1, 1983.
[19]: Papadimitrou C., Steiglitz K., "Combinatorial Optimization: Algorithms and Complexity", Prentice-Hall, Englewood Cliffs, N.J.,1982.
[20]: Tsang E. and Warwick T., "Applying Genetic Algorithms to Constraint Satisfaction Optimization Problems". Proceedings of the Ninth European Conference on Artificial Intelligence, 649-654, 1990
[21]: Whitley D., Starkweather T. and Fuquay D., "Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator", Proceedings of the Third International Conference on Genetic Algorithms, 133-140, 1989.
[22]: Widmer M. and Hertz A., "A new Approach for Solving the Flow Shop Sequencing Problem", ORWP 87/15, Department of Mathematics, University of Lausanne, 1987, to appear in European Journal of Operational Research.

Un algoritmo paralelo de descomposición LU para matrices 'sparse'

Maruja Ortega F. & Alfonso Reinoza
Dpto. de Computación y Tecnología de la Información
Universidad Simón Bolívar
Caracas, Venezuela
emsca!usb!mof@sun.com

Resumen: Se describe un algoritmo paralelo de descomposición LU para matrices 'sparse' no simétricas utilizando una arquitectura MIMD con memoria distribuida. El objetivo es encontrar una factorización de la matriz que minimice el llenado y que a la vez sea estable. Para reducir el llenado se utiliza el criterio de Markowitz. Las matrices se distribuyen de manera envolvente ('wrap-mapping'); la matriz triangular superior U por columnas y la matriz triangular inferior L por filas . Esta distribución es aprovechada para reducir el tiempo de selección del pivote en cada iteración y para acelerar el proceso de reducción.

Palabras Claves: MIMD, descomposición LU, matrices sparse, wrap-mapping.

1 Introducción.

Son muchas las aplicaciones que requieren la resolución de sistemas de ecuaciones lineales de la forma Ax=b de una manera estable. Una de las formas más comunes de llevar esto a cabo es mediante la descomposición de la matriz A en una matriz triangular superior U y una matriz triangular inferior L, con lo cual el sistema Ax=b se resuelve mediante la solución del sistema triangular inferior Ly=b seguido de la solución del sistema triangular superior Ux=y. Una de tales aplicaciones es el algoritmo Simplex. Una versión de este algoritmo, planteada por Bartels y Golub (1969)[3], está basada en la descomposición LU de la matriz básica B asociada a cada iteración.

Este trabajo en particular se enmarca dentro de un proyecto mayor [12] para el desarrollo de una versión paralela del algoritmo de Bartels y Golub, hecho que debió ser tomado en cuenta para la selección de las estructuras de datos así como de las rutinas de comunicación utilizadas.

El algoritmo presentado resuelve el problema de encontrar la descomposición LU de una matriz no simétrica y 'sparse' utilizando una arquitectura MIMD con memoria distribuida.

El objetivo es encontrar una factorización de la matriz que minimice el llenado y que a la vez sea estable, por lo que se debe llegar a un compromiso entre

estos dos criterios. Para resolver el problema del llenado se utilizó el criterio de Markowitz, empleado con éxito por Reid [11] y Gill, Murray, Saunders y Wright [8], en sus algoritmos secuenciales. En este sentido tambien fueron considerados los métodos de ordenamiento P3 y P4 de Hellerman y Rarick, utilizados por Saunders [15], los cuales fueron descartados por estar basados en recorridos tipo DFS de la matriz, siendo este último uno de los algoritmos hasta la fecha catalogados como "dificilmente paralelizables" [14].

La distribución de los datos se ha hecho de manera envolvente ('wrap-mapping'); por columnas para la matriz original B y para la matriz triangular superior U, mientras que la matriz triangular inferior L se distribuye por filas. Por cuanto se supone que la descomposición LU de una matriz será en general utilizada para la posterior solución de sistemas de ecuaciones, y los métodos que hasta la fecha ofrecen mejores resulados requieren que las matrices estén distribuidas de manera envolvente, se desea que las matrices triangulares resultantes preserven esta distribución. Una consecuencia de esto es que durante el proceso de factorización es necesario hacer el pivoteo explicitamente.

2 Algoritmo de Descomposición.

El algoritmo que se plantea recibe como información de entrada una matriz A mxn distribuida por columnas en forma envolvente y un vector indB que contiene los indices de las m columnas de A que deben conformar la matriz B a la que se aplicará la descomposición LU.

Los pasos del algoritmo ejecutados por cada procesador son los siguientes:

1. Inicialización de estructuras.
2. Para i=1 hasta m hacer
   - 2.1 Seleccion del pivote usando criterio de Markowitz.
   - 2.2 Pivotear filas y columnas.
   - 2.3 Actualizar estructuras.
   - 2.4 Reducción.

El primer paso corresponde a la inicialización de las estructuras correspondientes a las matrices L y U, así como la información necesaria para la seleción de los pivotes aplicando el criterio de Markowitz. La matriz B corresponde a la matriz inicial U. La matriz L inicial es la identidad.

2.1 Estructuras de Datos.

La matriz A se almacena por columnas utilizando tres arreglos unidimensionales StartPos_A, Val_A e IndRow_A, los cuales contienen la

posición donde se se encuentra el primer elemento de cada columna, los valores de los elementos y la fila que ocupa cada uno. Los elementos de una misma columna i se encuentran en posiciones contiguas desde StartPos_A[i] hasta StarPos_A[i+1]-1.

Para almacenar la matriz U se utiliza una estructura de listas por columnas, similar a la utilizada por Reid [14], implementada mediante cuatro arreglos unidimensionales StartPos_U, Val_U, IndRow_U y ColLink, que contienen la posición del primer elemento de cada columna, los valores de los elementos, la fila que ocupan y la posición del próximo elemento de la misma columna. Esta estructura de listas facilita el crecimiento del número de elementos de la matriz ya que no requiere que los elementos de una misma columna se encuentren en posiciones contiguas.

La matriz triangular inferior L será almacenada por filas, utilizando una estructura similar a la anterior.

La distribución de las matrices L y U por filas y por columnas respectivamente permitirá en un futuro comparar el comportamiento de los algoritmos correspondientes para resolver sistemas de ecuaciones triangulares. La matriz B se construye a partir de las columnas de A y del vector indB durante el proceso de inicialización y se almacena en el espacio reservado para U.

Parala aplicación del criterio de Markowiz se requiere información acerca del tamaño o cantidad de elementos distintos de cero de las filas y columnas de B, la cual se almacena en sendos arreglos de tamaño m durante el proceso de inicialización. Igualmente se dispone de una lista ElemsxCol que enlaza las columnas con igual cantidad de elementos organizándolas en orden creciente. Cada procesador dispondrá sólo de la información referente a su conjunto de columnas, mientras que la información de las filas deberá estar replicada en todos.

2.2 Aplicación del Criterio de Markowitz.

El criterio de Markowitz para matrices no simétricas y 'sparse' consiste en seleccionar como elemento pivote en una iteración k, un elemento $B_{ij}^{(k-1)}$ de la matriz reducida $B^{(k-1)}$, obtenida luego de k-1 iteraciones. El elemento seleccionado debe minimizar el producto $M_{ij} = (f_i-1)(c_j-1)$, donde f_i es el tamaño de la fila i y c_j es el tamaño de la columna j. Para que la factorización resultante sea estable se agrega una prueba de estabilidad con un margen de tolerancia μ, $0 < \mu \leq 1$, de forma que una vez hechas las permutaciones se cumpla que $|B_{kk}^{(k-1)}| \geq \mu\ |B_{ik}^{(k-1)}|$ para $k<i\leq m$.

Una aplicación estricta del criterio de Markowitz implica una revisión completa de la matriz reducida.

Para recortar esta búsqueda, en el algoritmo paralelo propuesto, cada procesador selecciona su candidato a pivote buscando sólo en sus q columnas con menos elementos y aplicando como estrategia para resolver empates entre dos elementos, quedarse con el primero de ellos que fue examinado. Luego se comparan los candidatos de los p procesadores para entre ellos seleccionar el mejor. En esta decisión basta tomar en cuanta el criterio de Markowitz pues todos los candidatos ya han pasado la prueba de estabilidad. Los empates se resuelven escogiendo el candidato que se encuentre en la columna con menor índice. Este criterio es definitivo puesto que no puede haber dos columnas con el mismo índice y garantiza que todos los procesadores seleccionarán el mismo elemento. Esto es imprescindible para mantener la consistencia de la información y el buen funcionamiento del algoritmo.

Comparando el comportamiento del algoritmo paralelo propuesto con su equivalente secuencial, se observa que si el mismo valor q que se utiliza en el algoritmo secuencial es utilizado en paralelo por cada uno de los procesadores, entonces el tiempo que tarda cada uno en seleccioner su candidato es el mismo que tarda el algoritmo secuencial en encontrar el pivote definitivo, a lo que debe agregarse, en el caso paralelo, el proceso posterior de elección entre los candidatos de cada procesador. Sin embargo, es de notar que para un valor de q, el resultado que se obtiene en paralelo es equivalente al obtenido en secuencial con q'=q*p. En vista de lo anterior y tomando en cuenta que el valor utlizado para q suele ser pequeño, bastará con fijar q=1 en el algoritmo paralelo para obtener resultados equivalentes a los que se pueden obtener en secuencial utilizando q=p.

2.3 Pivoteo y Actualización de Listas.

Dado que se debe mantener el balance de la carga de trabajo y la distribución envolvente será necesario hacer explicitamente el intercambio de la fila k con la fila pivote y de la columna k con la columna pivote. Sin embargo, las estructuras elegidas para el almacenamiento de L y U, permiten que las permutaciones de filas en U y las de columnas en L puedan hacerse de forma implícita, incorporándolas a las matrices de permutación P y Q.

En el proceso paralelo de intercambio de filas y columnas podrán estar involucrados hasta tres procesadores, a saber, el procesador que tiene la fila y la columna k y los procesadores que tienen la fila pivote y la columna pivote respectivamente. Notese que estos procesadores no necesariamente serán todos

distintos. En caso de haber al menos dos procesadores distintos, se establecerá entre ellos comunicación para hacer el intercambio.

La actualización de la información utilizada en la aplicación del criterio de Markowitz implica que cada procesador deberá disminuir en uno el tamaño de las columnas que tienen elementos en la fila del pivote, y tanto la fila como la columna del pivote deberán ser eliminadas de las listas de elegibles para futuras iteraciones.

Posteriormente, el cálculo de $B^{(k)}$ podrá generar nuevos elementos distintos de cero en algunas filas, en cuyo caso el tamaño de estas deberá ser nuevamente actualizado, al igual que el de las columnas correspondientes y, en consecuencia, la lista ElemsxCol.

2.4 Reducción de la matriz.

El proceso de reducción de la matriz $B^{(k-1)}$ para obtener $B^{(k)}$ comprende la eliminación de los elementos bajo la diagonal en la columna k mediante el cálculo de los multiplicadores, estos serán agregados a la fila correspondiente de L y utilizados para obtener $B^{(k)}$ sumando a cada fila i, $k<i\leq m$, el producto de la fila k por el multiplicador γ_{ik}.

Una alternativa para realizar este proceso es que el procesador que posee la columna k calcule uno a uno los multiplicadores correpondientes a las filas con elementos distintos de cero en esa columna. Cada multiplicador una vez que es calculado es inmediatamente enviado al resto de los procesadores para que actualicen sus columnas. El procesador que posee la fila del multiplicador, incluirá éste en L. El envío del multiplicador apenas es calculado permite que todos los procesadores comiencen a trabajar casi de inmediato.

Otra alternativa es que el procesador con la columna k calcule primero toda la columna de multiplicadores para luego enviarla al resto de los procesadores. Esta alternativa presenta la ventaja que sólo se establece comunicación una vez. La desventaja es que el cálculo del vector de multiplicadores se convierte un cuello de botella ya que durante este tiempo el resto de los procesadores estarían ociosos esperando. En algoritmos para matrices densas este inconveniente es posible reducirlo a un mínimo mediante una especie de encauzamiento en la aplicación del pivoteo parcial. En el caso 'sparse' esto no es posible pues en una iteración k no se sabe cual será columna pivote en la iteración k+1..

Entre estas alternativas la seleccionada fue la primera, pues el problema de serialización parece menor, a lo que se puede agregar que en teoría el tiempo de inicialización ('start-up') en los transputers es casi despreciable.

3 Resultados Computacionales.

El algoritmo presentado ha sido implementado en INMOS C paralelo para ser ejecutado sobre una red de p transputers conectados en anillo bidireccional. Se utilizó como plataforma de comunicaciones el subsistema correspondiente de SIMPAR[12], para el cual se require como mínimo p=3.

Se realizaron pruebas con p=1 y p=4 variando el tamaño y densidad de las matrices. Sin embargo, debido a la limitada capacidad de la configuración disponible no se pudo probar al comportamiento del algoritmo con matrices de grandes dimensiones. Las matrices utilizadas para las pruebas son bases optimas de problemas de progamación lineal.

En todos los casos, para la aplicación del criterio de Markowitz se fijo q=4 para el secuencial y q=1 para el paralelo, obteniendose siempre resultados similares o mejores en el paralelo.

Los resultados obtenidos muestran que para matrices de tamaño pequeño a medio, el balance entre tiempo de cómputo y tiempo de comunicaciones es desfavorable, de forma que con el algoritmo secuencial se obtienen resultados mejores que con el paralelo. Estos resultados se muestran en la tabla 1.

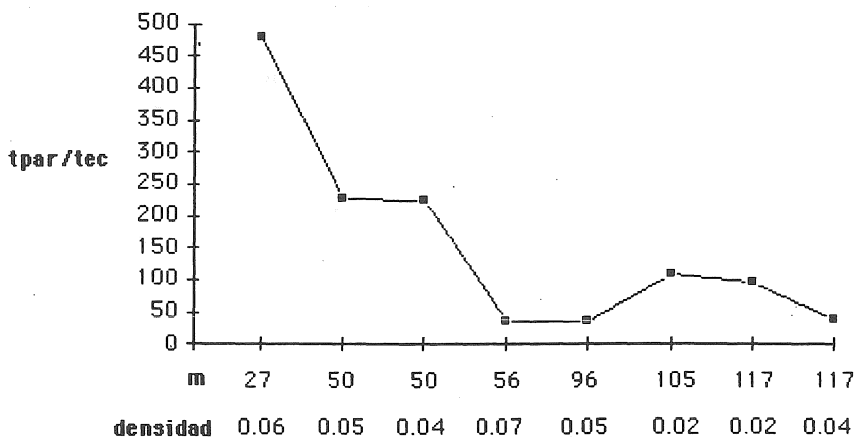


tabla 1.

A pesar de lo anterior, se observa que para un número constante de procesadores, la relación entre los tiempos de ejecución en paralelo y secuencial, tpar/tsec, se hace menor a medida que aumenta el tamaño de la matriz o la

densidad. Este resultado parece indicar que para matrices lo suficientemente grandes, la relación tpar/tsec debe invertirse haciendose más rápido el algoritmo paralelo. Esto último tiene sentido puesto que mientras más grande sea la matriz, mayor es el volumen de cálculo y en un determinado momento este volumen debe superar el over-head de comunicaciones necesario para poder repartir el trabajo. Sin embargo, deberán hacerse más pruebas para estudiar la veracidad de estas observaciones.

Por otro lado, debe notarse que al aumentar el número de procesadores, también debe ser mayor el tamaño de matriz para que el algoritmo paralelo pueda ser eficiente, debido al incremento de la distancia máxima entre procesadores lo cual influye en el tiempo de comunicaciones.

4 Conclusiones.

El algoritmo paralelo presentado resuelve el problema de encontrar la descomposición LU para matrices 'sparse' no simétricas aplicando el criterio de Markowitz para reducir el llenado. Las matrices triangulares resultantes quedan distribuidas de manera envolvente ('wrap-mapping') , lo que facilita la posterior aplicacion de algoritmos de resolución de sistemas de ecuaciones lineales. Las estructuras de almacenamiento elegidas para las matrices L y U se adaptan a requerimientos futuros, como lo es que esta descomposición pueda ser facilmente actualizada durante la aplicacion de la version de Bartels y Golub del algoritmo Simplex.

Para poder establecer el desempeño real del algoritmo se deberán hacer más pruebas aumenando el tamaño de las matrices.

Queda planteado estudiar el comportamiento del algoritmo si se agrega información adicional sobre la distribución de los elementos en las filas de B, o lo que es lo mismo, de U. Igualmente, dado el peso de las comunicaciones en al algoritmo, conviene volver a estudiar la alternativa de enviar en un solo mensaje toda la columna de multiplicadores.

Referencias:

[1] Gita Alaghband, H.F. Jordan. "Sparse Gaussian Elimination with Controlled Fill-in on a Shared Memory Multiprocessor." IEEE Trans. Comp. Vol. 38 Nº11, Nov 1989.

[2] R.H. Bartels. "A Stabilization of the Simplex Method." Numer. Math. 16, 1971.

[3] R.H. Bartels & G.H. Golub. "The Simplex Method of Linear Programming using LU Decomposition." Commun. ACM. 12, 1969.

[4] T.A. Davis & P. Yew. "A Nondeterministic Parallel Algorithm for General Unsymmetric Sparse LU Factorization". SIAM J. Matrix Anal. Appl. Vol11, Nº3. July 1990.

[5] I.S. Duff, A.M. Erisman & J.K. Reid. "Direct Methods for Sparse Matrices." Clarendon Press. Oxford. 1986.

[6] J.J.H. Forrest & J.A. Tomlin. "Updating Triangular Factors of the Basis to Maintain Sparsity in the Product Form Simplex Method. " Math. Programming 2, 1972.

[7] G.A. Geist, C.A. Romine. "LU Factorization Algorithms on Distributed Memory Multiprocessor Architectures." SIAM J. Sci. Stat. Comput. Vol 9. Nº4. July 1988.

[8] P.E. Gill, W. Murray, M.A. Saunders & M.H. Wright. "Maintaining LU Factors of a General Sparse Matrix." Linear Algebra Appl. 88/89, 1987.

[9] M.T. Heath, C.H. Romine. "Parallel Solution of Triangular Systems on Distributed-memory Multiprocessors" Tech. Report ORNL/TM-10384, Oak Ridge National Laboratory, Oak Ridge, TN. 1987.

[10] Guangye Li, T.F. Coleman "A Parallel Triangular Solver for a Distributed Memory Multiprocesor." SIAM J. Sci. Stat. Comp. Vol. 9, Nº 3, Mayo 1988.

[11] J.K. Reid. "A Sparsity-exploiting Variant of tha Bartels-Golub Decomposition for Linear Programming Bases." Math. Programming 24, 1982.

[12] A. Reinoza, M. Lentini, A. Teruel, L. Da Silva, A. Guillen, T. Lozada, O. Naím. "Simplex Paralelo (SIMPAR). Fase 1." Proceedings PANEL'91, XVII Conferencia Latinoamericana de Informática, Vol II. 991-1008.

[13] A. Reinoza et al. "Simplx Paralelo (SIMPAR). Fase 2. Reporte Interno. Dpto. de Computación. Universidad Simón Bolívar. Feb 1992.

[14] W. Rytter & A. Gibbons. "Efficient Parallel Algorithms". Cambridge Univ. Press. 1988.

[15] M.A. Saunders. "A Fast, Stable Implementation of the Simplex Method using Bartels-Golub Updating." Sparse Matrix Computations. Rose & Bunch, Eds. 1976.

Trabalho cooperativo e correio eletrônico: uma experiência real no projeto ProTem-CNPq.

José Palazzo M. de Oliveira
Universidade Federal do Rio Grande do Sul
Instituto de Informática (1)

Norberto Hoppen
Universidade Federal do Rio Grande do Sul
Programa de Pós-graduação em Administração

Eratóstenes E. Ramalho de Araujo
CNPq

Resumo

A coordenação da elaboração de um projeto de pesquisa em Ciência da Computação, envolvendo um conjunto significativo de organizações de pesquisa e cerca de 1200 pessoas distribuídas em todo o território do Brasil, apresentou uma complexidade muito grande. A utilização do correio eletrônico como instrumento de coordenação foi um dos fatores que contribuíram para o sucesso desta experiência. Este artigo descreve o projeto ProTem e analisa a utilização do correio eletrônico, nos aspectos de coordenação e de homogeneização dos pontos de vista dos participantes, características, estas, essenciais para um trabalho cooperativo desta extensão.

1. Introdução

O crescimento da complexidade do gerenciamento das atividades das organizações tem provocado um uso crescente do trabalho em grupo. Este tipo de trabalho tem como vantagens a obtenção de soluções mais ricas e completas para a problemática tratada mas, por outro lado, requer atividades adicionais de organização para manter a eficácia do grupo [Hub 80].

---

1 Av. Bento Gonçalves, 9500 - Bloco IV - Agronomia - Caixa Postal 15064 - CEP 91501 - Porto Alegre - RS - Brasil. Fax +55(51)336.5576, e-mail: palazzo@inf.ufrgs.br

O correio eletrônico é um serviço que permite a troca de informações, através de mensagens, entre pessoas dispersas geograficamente, via uma rede de computadores. É um dos suportes tecnológicos que pode facilitar a circulação das informações e a coordenação do grupo. A tecnologia do correio eletrônico está disponível desde o início dos anos 70 mas somente foi implantada no Brasil em 1988.

Este estudo apresenta uma análise do uso do correio eletrônico em um projeto desenvolvido em grupo, denominado trabalho cooperativo [Ell 91], de grande envergadura - 55 instituições dispersas em todo território nacional -. Foram detalhados fatores ligados à coordenação do grupo e à troca de informações entre os participantes do mesmo. Para tanto, o artigo apresenta, inicalmente, o contexto do trabalho, o projeto *ProTem*, conceitua o trabalho cooperativo e analisa dados sobre coordenação -forma e conteúdo- e sobre troca de informações -origem e destino das mensagens, estrutura assumida pelo grupo.

2. O Programa ProTeM-CC / CNPq

Um dos principais órgãos de apoio à pesquisa no Brasil é o CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico. A forma de apoio, tradicionalmente, tem sido orientada ao financiamento de pesquisas a nível individual. O racional desta forma de ação consiste em supor que o fator determinante da qualidade da pesquisa consiste na competência individual do pesquisador.

Dentro desta forma de trabalho, o CNPq publica bianualmente um calendário de datas para a submissão de pedidos de auxílio. Os pedidos recebidos são enviados para a avaliação por especialistas na área, os pareceres técnicos são recebidos e o conjunto de pedidos de auxílio com parecer favorável são avaliados em reunião específica pelo Comitê Assessor da respectiva área de conhecimento.

Nos últimos anos, a consolidação de um certo número de grupos de pesquisa em Ciência da Computação tem elevado constantemente o grau de qualidade científica dos projetos apresentados. Por outro lado, a forma de apoio tradicional do CNPq estimula, de certa forma, um isolamento dos pesquisadores em seus projetos. Assim, hoje, existem grupos consolidados de pesquisa espalhados pelo país e projetos, na sua maior parte, realizados em apenas uma instituição. Além disto, a grande quantidade de pedidos individuais de auxílio à pesquisa levam a um acúmulo exagerado de trabalho administrativo, com dificuldades claras para as atividades de acompanhamento e avaliação.

A situação dos grupos emergentes, nesta estrutura, é relativamente difícil. Estes grupos, que contam com uma pequena infra-estrutura e com poucos pesquisadores, tem encontrado dificuldades em competir com os grupos consolidados na busca de recursos.

No final do ano de 1989 o CNPq, em contraposição à sua forma tradicional de ação, iniciou um novo e inovador projeto na área de Ciência da Computação: o Projeto Temático Multiinstitucional em Ciência da Computação - *ProTem-CC*. Os objetivos principais deste projeto foram estipulados como:

- mudar decisivamente o status da pesquisa e da formação de recursos humanos em ciência da computação no país;

- promover efetivamente a integração multiinstitucional através da cooperação entre grupos de pesquisa (consolidados e emergentes) em torno de temas estratégicos de interesse nacional;

- executar uma centena de projetos de pesquisa nos temas estratégicos, diretamente apoiados e acompanhados no contexto dos objetivos do programa;

- articular esforços de cooperação bilateral em pesquisa pré-competitiva com os principais países e programas supranacionais;

- implantar um processo sistemático de planejamento e execução de atividades.

2.1 O ambiente de trabalho

Para controlar a complexidade da tarefa de formulação de projetos do programa *ProTem,* e posteriormente, ter facilitada a sua administraçãobem como para reduzir os custos associados ao deslocamento de pesquisadores, a apresentação dos projetos de pesquisa ficou sob a responsabilidade de cinco coordenações regionais[2]. Estas cinco coordenações abrangem as seguintes regiões: Nordeste, Centro-oeste, Rio de Janeiro, São Paulo e Sul (tabela 1). Em cada região uma instituição sediou a coordenação regional, assumindo as atividades de coordenação e administrativas, através de um coordenador regional.

Regiões	Coordenação	Grupos
Nordeste	UFPE	5
Centro-Oeste	UFMG	6
Rio de Janeiro	UFF, IMPA	6
São Paulo	UNICAMP	6
Sul	UFRGS	5

Tabela 1: Regiões e número de grupos inicialmente envolvidos

O programa *ProTem*, como todos os projetos de pesquisa no âmbito do CNPq, se desenvolve em cinco fases: elaboração, julgamento, execução, acompanhamento e avaliação. Este artigo trata da primeira fase: a elaboração do projeto, a qual durou oito meses. As atividades desenvolvidas foram dedicados à elaboração das propostas de projetos multiinstitucionais a nível de região (tabela 2).

---

[2] Dadas as distâncias encontradas no Brasil a maior parte dos grupos encontram-se afastados de centenas ou mesmo milhares de quilômetros. A maior exceção constitui-se a região de São Paulo onde as distâncias entre os principais grupos estão limitadas a cerca de 200 km e no Rio de Janeiro onde os grupos são locais.

No período anterior a março foram tomadas as decisões políticas a nível do CNPq para a definição das diretrizes e montagem do projeto. A seguir foram indicados os coordenadores regionais.

Em 15 de março de 1991 foi realizada a primeira reunião, na sede do CNPq, entre os coordenadores regionais, membros do CA-CC e pessoal técnico do CNPq. Nesta reunião ficou evidenciada a pouca experiência existente para o desenvolvimento de um projeto deste porte[3].

Datas	Atividades	Locais
< março	definição diretrizes projeto	Brasília
15 mar 91	reunião inicial no CNPq	Brasília
6-7 jun 91	reunião de coordenação	Rio
ago 91	entrega resumos executivos	regiões
nov 91	projeto finalizado	regiões

Tabela 2: Calendário de eventos

Como primeira providência, para acelerar a acumulação de conhecimentos e agilizar o processo decisório, foi proposta a instalação de uma lista de discussão, via correio eletrônico. Uma lista é constituída por um conjunto de endereços de correio eletrônico dos participantes e por seu próprio endereço - o nome da lista. Qualquer participante pode enviar uma mensagem para a lista mensagem que, uma vez recebida pelo software gerenciador da lista, é automaticamente retransmitida para todos os membros da lista. Desta forma é possível manter uma discussão, assíncrona, entre todos os membros, o que facilita a coordenação. Nesta lista - PROTEM-L - estão cadastradas as treze pessoas referenciadas acima.

Esta ferramenta permitiu suportar a troca eficiente de opiniões, a participação no planejamento de membros dispersos nacionalmente e a resolução de problemas específicos durante a elaboração do projeto.

Nos meses de abril e maio foram iniciados os trabalhos relativos aos grupos temáticos regionais. As instituições indicaram os seus representantes, elas foram agrupadas por regiões e, nestes grupos, realizaram-se reuniões de apresentação do projeto. Como resultado destas reuniões, os diferentes tópicos de interesse das instituições foram identificados bem como criados grupos de trabalho para coordenar os subprojetos regionais. A nível regional, um fato importante foi a instalação de listas abertas aos pesquisadores das regiões para a discussão dos tópicos locais. Desta forma as decisões tomadas pela comissão de coordenação do projeto nacional puderam ser rapidamente repassadas e discutidas entre todos os participantes. No caso da região sul foram trocadas 293 mensagens entre os participantes.

---

[3] Inicialmente estavam previstas a participação de cerca de 213 doutores, 103 mestres; ainda estariam vinculados a estes pesquisadores cerca de 80 alunos de doutorado, 300 alunos de mestrado e 450 bolsistas de iniciação científica.

A seguir o CNPq forneceu um conjunto de estações de trabalho SUN para suprir adequadamente os diferentes grupos envolvidos[4]. Estes equipamentos objetivam garantir as condições mínimas de desenvolvimento dos grupos emergentes de forma a poderem suportar as pesquisas propostas. Este fato demonstrou, para as instituições envolvidas, o real engajamento do CNPq no projeto.

Nos dias 6 e 7 de junho foi realizada uma reunião com os participantes da coordenação nacional para o estabelecimento de regras e padrões para a homogeneização dos diferentes projetos regionais. Nesta reunião cada coordenador apresentou o estado dos sub-projetos regionais bem como as necessidades de recursos para seu desenvolvimento. A seguir foi realizada uma pré-avaliação das necessidades e recursos envolvidos para cada região.

No final de agosto foram entregues ao CNPq os resumos executivos de cada região para a elaboração do orçamento de 1992. A seguir cada coordenador regional concluiu a elaboração dos projetos e enviou-os ao CNPq para o processo de avaliação. No total foram consumidas 32 semanas, desde a reunião inicial em Brasília até a conclusão dos projetos regionais.

2.2 O grupo de trabalho

A nível nacional, 13 pessoas participaram ativamente na elaboração do projeto. Destas, 5 foram os coordenadores regionais e mais 2 colaboradores[5], 4 são membros do corpo técnico do CNPq e finalmente 3 são membros do CA-CC. Todos os pesquisadores são doutores com um histórico de atividades em ensino, pesquisa e administração de pesquisa.

Os participantes deste grupo de trabalho podem ser caracterizado como "early adopters" [Gur 90], isto é, pessoas que aceitam bem a inovação tecnológica. A quase totalidade dos participantes do grupo, tão logo se tornou razoavelmente operacional o correio eletrônico, foram dos primeiros a utilizar esta tecnologia no país. Assim , o grupo possui um bom domínio da tecnologia e uma visão positiva sobre os benefícios das redes de comunicação. Ele pode ser considerado como dotado de uma atitude positiva de cooperação.

Como as diferentes regiões não estavam competindo por recursos[6], a cooperação pôde desenvolver-se de forma adequada.

---

[4] A seleção dos grupos a serem aquinhoados foi realizada pelo CA-CC baseado na qualidade científica dos grupos e, para os grupos emergentes, no esforço de capacitação docente realizado pelos mesmos (professores em programa de doutoramento bem como as datas de retorno previstas).

[5] Estes dois colaboradores extra trabalharam na região do Rio de Janeiro devido a complexidade particular deste projeto.

[6] Uma das proposições do *ProTem* é a de realizar um levantamento real das necessidades de apoio à pesquisa.

3. Trabalho cooperativo

O trabalho de múltiplos atores, visando um objetivo comum, requer uma série de atividades para a sua organização do trabalho. Estas atividades adicionais de organização são denominadas *coordenação*. Segundo Malone [Mal 88], a coordenação implica nos seguintes componentes: (1) dois ou mais atores, (2) que desenvolvem atividades, (3) para atingir um determinado objetivo.

No caso em estudo os atores são representados pelos membros do grupo responsável pelo projeto nacional. As atividades desenvolvidas foram: a organização do grupo, a instalação dos grupos temáticos regionais, a coordenação dos trabalhos de elaboração dos projetos regionais, a homogeneização dos projetos e elaboração dos resumos executivos, e a montagem dos documentos finais. O objetivo a ser atingido foi a elaboração de um projeto nacional englobando o maior número possível de pesquisadores em Ciência da Computação.

Para tornar possível a coordenação é importante o emprego de *tecnologias de coordenação*[Mal 88], tais como correio eletrônico, teleconferência e agenda de grupo. Neste caso a tecnologia de coordenação foi, basicamente, a comunicação via computadores. Esta tecnologia permite a formação de grupos com os participantes geograficamente dispersos e sem necessidade de simultaniedade nas reuniões.

A comunicação por meio de computadores tem por objetivo estabelecer a troca de informações entre indivíduos ou grupos. Como produto dessas trocas, geralmente surgem vários tipos de mensagens e outros objetos (tais como documentos ou tabelas) para concretizar a comunicação.

As diferentes sedes das coordenações regionais, bem como o CNPq, estão ligadas por meio de circuitos TRANSDATA[7] da EMBRATEL. Os computadores dos diferentes grupos de pesquisa estão interligados como componentes da rede INTERNET.

4. Análise dos dados

O objetivo deste estudo é a ligação da tecnologia de coordenação empregada com a resolução do problema: a geração de um projeto nacional na área de ciência da computação.

As mensagens transmitidas para a lista, ao longo das 32 semanas de trabalho, foram armazenadas e analisadas segundo os seguintes critérios de classificação: freqüência das mensagens por origem, origem e destino das mensagens e categorias de mensagens. O propósito desta classificação foi obter uma análise do tipo de comunicações entre os participantes e da contribuição do correio eletrônico para a coordenação do mesmo.

---

[7] Circuitos dedicados para a comunicação de dados.

Foram emitidas 168 mensagens, sendo que 63 com origem no CNPq e as restantes 105 originárias dos coordenadores. Dos participantes da lista dois técnicos do CNPq e o coordenador do Ca-CC não emitiram mensagens. Em entrevistas realizadas posteriormente, informaram que utilizaram as mensagens recebidas como fonte de informações para suportar as suas atividades.

Na tabela 3 constata-se que São Paulo, em função do maior número de instituições, gerou um volume maior de mensagens. Por outo lado, em São Paulo também ocorreram o maior número de problemas com a rede física (mais de 22 mensagens). A reunião geral de coordenação foi realizada no Rio de Janeiro, fazendo com que um grande número de problemas já detectados em outras regiões fossem resolvidos localmente.

Número de mensagens	Número de instituições	Origem
46	15	São Paulo
29	11	Sul
17	9	Centro Oeste
9	13	Rio de Janeiro
4	7	Nordeste

Tabela 3: Freqüência das mensagens por origem

Na tabela 4 estão apresentadas as freqüências das mensagens, considerando a origem e o destino das mesmas. As colunas *Coordenação CNPq* e *Coordenadores regionais* indicam as mensagens que possuíam um destinatário específico, se bem que todas as mensagens foram recebidas por todos os participantes da lista. As categorias, representadas pelas colunas não são exclusivas, pois uma mensagem pode ter parte dirigida a todos os participantes e parte dirigida a um participante específico.

PARA DE	Todos	Coordenação CNPq	Coordenadores regionais
Coordenação CNPq	48	--	15
Coordenadores regionais	76	15	17

Tabela 4: Origem e destino das mensagens

A partir da freqüência registrada na primeira coluna pode-se concluir que a coordenação do projeto envolveu tanto a comunicação *top-down* da coordenação central, a partir do CNPq, em um total de 63 mensagens, quanto a comunicação horizontal entre os coordenadores regionais, representada por 76 mensagens. Ocorreu, também, uma parcela de comunicação dos coordenadores regionais para o CNPq, 15 mensagens. Em termos globais observa-se uma maior percentagem, 72,5% de mensagens gerais, para todos os integrantes da rede, contra apenas 27,5% de mensagens específicas dirigidas para um participante.

A estrutura das classes de mensagens (tabela 5) depende do tipo de tarefa a ser realizada, neste caso a elaboração de um projeto de pesquisa. Especificamente, para a tarefa de elaboração do projeto, depende de uma estrutura organizacional própria, dos objetivos e estratégias adotados, das atividades definidas e dos recursos disponíveis.

Classes não exclusivas	mensagens
Organização de reuniões	36
Envio de respostas	27
Opiniões/posicionamentos	26
Difusão de informações	24
Rede (suporte tecnológico)	22
Pedido de informações dos coordenadores	21
Coordenação do projeto	20
Distribuição de relatórios	12
Pedido de informações ao CNPq	10
Elaboração de documentos	10
TOTAL	208

Tabela 5: Classes de mensagens

A análise das classes das mensagens permitem estudar o processo de elaboração do projeto como um todo. Este processo foi fortemente influenciado pela existência da lista de discussão devido à instantaneidade da comunicação e pela disponibilidade imediata da informação por todos os coordenadores regionais.

As classes vinculadas à coordenação: "Envio de respostas", "Pedido de informações dos coordenadores", "Coordenação do projeto" e "Pedido de informações ao CNPq" chegaram a 78 mensagens (37,5% do total)

A uniformização das opiniões e a obtenção de consenso entre os coordenadores foi facilitada pelas classes de mensagens "Opiniões/posicionamentos" e "Difusão de informações" em um total de 50 mensagens (24% do total).

As mensagens "Organização de reuniões" correspondem a 17,3% do total. O bom número de mensagens nesta classe garantiram uma homogenização prévia dos conteúdos dos subprojetos. Outro resultado positivo deste intercâmbio foi a possibilidade de organizar uma agenda relevante para a discussão. Isto, junto à uniformização das opiniões (41,3% das mensagens), permitiu a realização de apenas uma reunião de coordenação durante o transcorrer da elaboração do projeto, excluída a reunião preparatória inicial.

O número, relativamente grande, de mensagens referentes à tecnologia de coordenação ("Rede") deveu-se a falhas no "mail server" de São Paulo e nos canais de comunicação, muitas vezes sobrecarregados, entre os diversos computadores. Estes problemas estão sendo sanados pela implantação de uma Rede Nacional de Pesquisa - RNP - pelo CNPq que deverá suportar comunicações de boa confiabilidade entre os centros de pesquisa.

Finalmente, o intercâmbio de documentos, representado pela classe "Elaboração de documentos", permitiu a homogeneidade dos documentos na elaboração da proposta de orçamento.

5. Conclusão

A elaboração do projeto final resultou na participação de 55 instituições de pesquisa e cerca de 1200 pessoas. A base do ProTem está predominantemente localizada nas Universidades. Uma das mais importantes conseqüências desta primeira fase foi a identificação de diversos grupos emergentes e a interação destes grupos com os já consolidados. Diversos estágios de pesquisadores em outras instituições e trabalhos conjuntos estão em andamento, mesmo antes da alocação de recursos para os projetos propostos. O processo de "alavancagem" dos grupos emergentes é uma realidade.

Verificou-se também a emergência do sentimento, por parte dos participantes, de pertencerem a um grupo de interesse, pois 7% do total do conteúdo de mensagens referiu-se à troca de informações sobre assuntos acadêmicos não diretamente relacionados com o projeto.

A presente experiência foi desenvolvida em ambiente acadêmico. Algumas *parcerias* foram encontradas entre empresas e universidades durante o desenrolar do projeto, nas áreas de Engenharia de Software (nas cinco regiões), Informática Industrial (Sul), Otimização e Heurísticas Baseadas em Conhecimento (Centro-Oeste). Esta cooperação com empresas foi identificada em uma etapa pré-competitiva, quando são desenvolvidas tecnologias básicas e não produtos. Neste sentido o trabalho cooperativo pode facilitar a integração do ambiente acadêmico com projetos de interesse comercial.

Como principal ponto negativo foi identificada uma significativa variação na granularidade dos temas propostos pelas diferentes regiões. Este fato foi percebido na reunião de coordenação no Rio mas, neste momento, já era difícil a reformulação total dos subprojetos. Provavelmente este problema poderia ter sido sanado pela utilização de uma etapa de convergência nos temas. como tecnologia para esta etapa poderia ser empregada a técnica de Delphi.

Ficou claro, através desta experiência, que o emprego de tecnologia de coordenação baseada em comunicação via computadores permitiu que um grupo cooperativo atingisse um objetivo complexo em um tempo aceitável e com custos reduzidos. Esta experiência pode servir para embasar projetos mais amplos integrando a comunidade íbero-americana. Estes projetos tem encontrado dificuldades para sua realização devido aos custos do processo de coordenação e, também, devido a grande demora em se utilizar o correio normal.

6. Bibliografia

[Ell 91] C. A. Ellis et al, "Groupware: some issues and experiences", Communications of the ACM, v 34, n 1, Jan 91, p. 38-57.

[Gur 90] Vijay Gurbaxani, "Diffusion in Computing Networks: The case of BITNET", Communications of the ACM, v 33, n 12, Dec 90, p. 65-75.

[Hub 80] George P. Huber, "Managerial Decision Making", Glenview, Ill.: Scott, Foresman and Co., 1980, 228p.

[Kli 91] Rob Kling, "Cooperation, Coordination and Control in Computer Supported Work", Communications of the ACM, V 34, n 12, Dec 91, p. 83-88.

[Mal 88] Thomas W. Malone, "What Is Coordination Theory?", CISR WP Nº 182, Feb 88, Center for Information Systems Research, Massachusetts Institute of Technology.

Evaluación del Desempeño de sistemas Unix*en ambiente distribuido

Eduardo Pavez F.
Depto. de Cs. de la Computación
Universidad de Chile
epavez@uchdcc.uchile.cl

Gonzalo Urrea R.
Depto. de Cs. de la Computación
Universidad de Chile
gurrea@uchdcc.uchile.cl

Resumen

El sistema operativo Unix es uno de los más utilizados hoy en día, ya sea en centros de investigación o en las distintas actividades de las empresas. Sin embargo, no existe una buena difusión de técnicas de modelamiento de Unix que permitan evaluar su desempeño y aprovechar mejor sus características de administración y asignación de recursos.

Este trabajo consiste en el desarrollo de un modelo para evaluación del desempeño de sistemas que ejecutan en el sistema operativo Unix en un ambiente distribuido, basándose en los conceptos bajos los cuales se rige el modelo construido por Ramamurthy [13].

Los resultados obtenidos en la validación, del modelo de Ramamurthy, en un ambiente local y distribuido de multiprogramación difieren con respecto a los resultados reales en un 10%.

Por otra parte, es necesario destacar que la principal desventaja del modelo es el costo computacional de los algoritmos para evaluación de medidas de rendimiento, los que dependen del número de tipos de trabajos distintos, del número de trabajos por tipo, número de dispositivos modelados y otros parámetros que aumentan tanto los requerimientos de memoria como los de tiempo de ejecución del algoritmo.

keywords: modelamiento, evaluación del rendimiento, sistemas computacionales, sistemas distribuidos, sistema operativo Unix.

1 Introducción

El sistema operativo Unix es uno de los más utilizados en la actualidad, debido a la versatilidad y amplitud de su aplicación. Una de sus características principales es que es un sistema de

*Unix es marca registrada de Bell Laboratories

tiempo compartido que permite a varios procesos estar ejecutando simultáneamente, los que pueden tener distintos objetivos, por ejemplo: planificación de procesos, manejo de comandos del sistema operativo, edición de texto, consultas a bases de datos y en la actualidad acceso a redes de computadores.

Es posible que configuraciones basadas en Unix puedan sufrir problemas de desempeño y requerir modificaciones, esto genera interrogantes acerca de la utilización de los distintos dispositivos y las cargas de trabajo que provocan tales situaciones.

Para poder predecir los problemas y analizar las características de sistemas que ejecutan en Unix en un ambiente local o distribuido, deben existir herramientas que permitan realizar el registro de información del sistema, estas herramientas son conocidas como monitores para evaluación del desempeño [4][8].

La información entregada por los monitores junto a la información de hardware (ucp, discos, terminales, controladores de discos) y conocimiento del tipo de trabajos que ejecutan en el sistema, ha permitido y permitirá construir modelos que representen adecuadamente las características de Unix. Existen algunos estudios de Unix, los que han sido desarrollados por [10][11][9] y, uno de los más sofisticados, el modelo de Ramamurthy[13] sobre el cual está basada esta investigación.

2 Descripción de Unix para el diseño de un modelo distribuido

El sistema operativo Unix fue desarrollado por Ken Thompson, y luego reescrito en el lenguaje C por Dennis Ritchie [6], para laboratorios Bell. Con el aumento en desarrollo y poder de los microprocesadores, otras compañías llevaron el sistema Unix a nuevas máquinas, y por su simplicidad y claridad llevó a muchos ingenieros de sistemas a adaptarlo a sus propias necesidades, resultando varias versiones del sistema original.

Como principal característica es que permite a varios procesos ejecutar en forma simultánea, permitiendo a varios usuarios estar conectados a la vez. Es Unix el encargado de administrar la ejecución de los distintos procesos, mediante un esquema de tiempo compartido y asignación de prioridades [1].

En la Figura 1, se muestra la arquitectura lógica del sistema operativo Unix. El sistema operativo interactúa directamente con el hardware, entregando servicios a los procesos que ejecutan en él. Entonces, podemos ver al sistema operativo como un conjunto de componentes [1], en que la principal es conocida como el *kernel* del sistema, y tiene como función establecer la comunicación con el hardware.

La labor principal de Unix es el control de los recursos via la ejecución de llamadas al sistema. Los programas mostrados en la Figura 1, como GREP o VI, interactúan con el *kernel* a través de la ejecución de un conjunto de llamadas al sistema, conocidas como *system calls*. Estas llamadas al sistema hacen variadas operaciones a través de llamadas a rutinas proporcionadas por el sistema operativo, y por medio de estas se intercambia

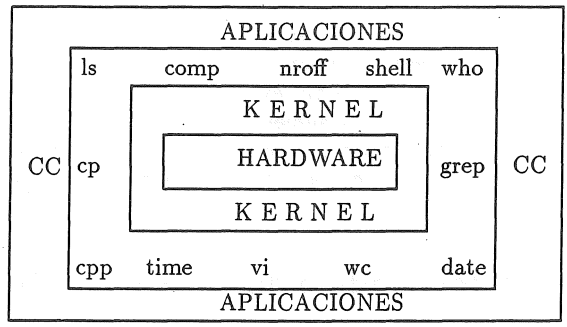


Figura 1. Arquitectura lógica del sistema operativo Unix

información entre el *kernel* y el programa del usuario. En definitiva, un programa en ejecución se transforma en un conjunto de llamadas al sistema junto al código efectivo del programa del usuario.

Por otra parte, el *kernel* se encarga de otras labores como asignar la UCP a los procesos, controlar la creación, suspensión comunicación y finalización de procesos, asignar memoria principal para procesos en ejecución, junto con la protección de áreas restringidas, administración de memoria secundaria, lo que incluye además el control y administración del sistema de archivos.

Este mismo esquema de Unix es posible ampliarlo a un sistema distribuido, como por ejemplo, si un comando antes ejecutaba una lectura en un disco local, ahora, en un esquema distribuido es probable que esta lectura se realice en forma remota, mediante alguna llamada al sistema dedicada a esa operación.

El modelo para Unix en ambiente distribuido, desarrollado en esta investigación, está basado en los recursos que requiere un proceso o comando para ser ejecutado en el sistema distribuido. Las definiciones utilizadas para sistemas distribuidos están basadas principalmente en las implementaciones para Unix descritas en [1].

Se pudo establecer en la práctica que la principal diferencia con un sistema local de multiprogramación, basado en Unix, radica en el lugar donde se atienden ciertos requerimientos, que no necesariamente los entrega la máquina donde se ejecuta el proceso. De acuerdo a esta característica, un proceso que no pueda satisfacer localmente una necesidad, realizando llamadas al sistema local, las efectuará en forma remota, y estas llamadas remotas al sistema son provistas por Unix.

El hecho que exista un servidor de archivos implica que el sistema tiene que ejecutar

procesos que le permitan enviar el requerimiento a través de la red, siendo atendido en el servidor y luego devuelto al cliente con la respuesta. El recorrido a través de la red produce un aumento en el tiempo de respuesta, comparado con un ambiente local. También, existe un aumento del tiempo de respuesta debido a que existirán una gran cantidad de requerimientos remotos sobre el servidor, producto de un aumento significativo de la carga. Si en el servidor la carga es baja, el cliente notará un tiempo de respuesta equivalente a como si el sistema atendiera el pedido localmente, y, por el contrario si la carga es alta el tiempo de respuesta será aún mayor.

3 Descripción del Modelo de Unix en ambiente distribuido

El modelo de Unix de Ramamurthy está construido en base a un Modelo de Red de Colas con Prioridades [5], que permite representar a un sistema computacional ejecutando en el sistema operativo Unix.

Cada sistema computacional en el sistema distribuido puede tener discos locales o hacer uso de un sistema de discos existente en otro computador, ver Figura 2 y 3.

Los procesos, ejecutando en alguno de estos sistemas, son representados mediante un esquema de múltiples clases y prioridades, caracterizadas por las llamadas al sistema y la propia ejecución.

Los usuarios son modelados como unidades que envían un proceso a ejecución, y tienen asociado un tiempo de reflexión, que corresponde al tiempo que se demora un usuario en enviar el siguiente comando.

Los discos corresponden a unidades que atienden procesos, en orden de llegada, y que son caracterizados por un tiempo de servicio por operación realizada en el disco y el número de veces que un proceso hace uso de ese disco.

La UCP está caracterizada como una unidad que atiende procesos de acuerdo a un esquema de prioridades, cuyos parámetros son el tiempo de servicio o *time slice* en Unix (fijo para todos los procesos) y el número de veces que un proceso visita la UCP. El esquema de prioridades es el siguiente:

- prioridad 1 y 2 son procesos del *kernel* ejecutando en modo *kernel*, y no son desalojables por otros procesos.
- prioridad 3 hasta K-1, son procesos ejecutando en modo *kernel* con prioridad a nivel de usuario y son desalojables por los procesos de prioridad 1 y 2.
- prioridad K, son procesos de usuario ejecutando a nivel usuario y son desalojables por procesos de mayor prioridad.

En este esquema de prioridades un menor valor de prioridad representa una prioridad mayor de atención, frente a otros procesos.

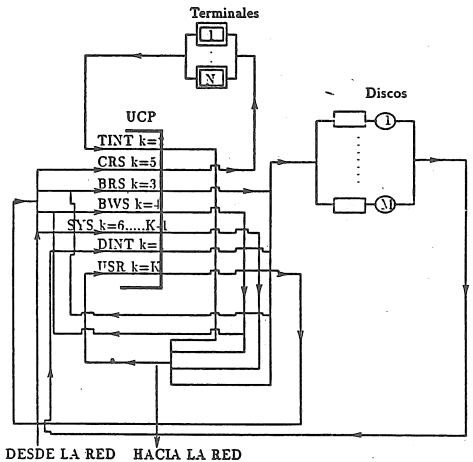


Figura 3. Modelo de Unix para un servidor de la red

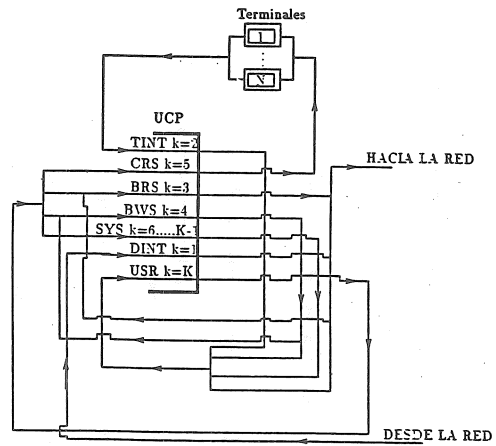


Figura 2. Modelo de Unix para un sistema sin discos

Inicialmente, un proceso que empieza su ejecución activa un interrupción asociada al terminal, con prioridad 2, llamada TINT. Luego, comenzará su ejecución realizando llamadas al sistema, las que corresponden a:

- DINT, interrupción desde el sistema de discos, indicando que el bloque requerido ha sido transferido al *buffer cache* de disco, con pioridad 1.
- CRS, lectura desde un terminal, con prioridad 5.
- BRS, llamada al sistema para escribir de disco, con prioridad 3 (*Block Read*), local y remota.
- BWS, escrituras a disco, con prioridad 4 (*Block Write*), local y remota.
- SYS, otras llamadas a rutinas de I/O con prioridad 6 a K-1. Estas pueden ser: *access*, *fork*, *lseek*, *open* o *wait*, entre otras.
- USR, ejecución a nivel usuario con prioridad K.

La principal característica, de acuerdo a la descripción anterior, es que el modelo de Ramamurthy representa a un tipo o clase de trabajo por sus diferentes llamadas al sistema.

Esta representación permitió a Ramamurthy desarrollar un conjunto de fórmulas analíticas basadas en prioridades [5][7] que permiten evaluar las medidas de desempeño, ya sea tiempo de respuesta por clase, throughput o utilización de distintos dispositivos. Algunas de las definiciones y fórmulas del modelo de Unix en ambiente distribuido se presentan a continuación:

r = caracteriza un tipo de trabajo (editor, consulta, transacción).

e = número de nodo en la red, de un total de E nodos o máquinas Unix.

c = dispositivo en un nodo (ucp, disco, cinta).

p = prioridad de ejecución en Unix.

N = número trabajos en el sistema = $\sum_{e=1}^{E} \sum_{r=1}^{R} N_{e,r}$

N_e = número de trabajos en el nodo e.

$S_{e,c,r,p}$ = tiempo de servicio para un trabajo de tipo r ejecutando con prioridad p, en el dispositivo c del nodo e.

$V_{e,c,r,p}$ = visitas que realiza un trabajo de tipo r ejecutando con prioridad p, en el dispositivo c del nodo e.

$\rho^r_{e,c,r,p}(N_e)$ = probabilidad que un trabajo de tipo r encuentre a un trabajo de tipo m ejecutando con prioridad p, en el dispositivo c del nodo e, cunado hay N_e trabajos en el nodo e.

$L_{e,c,r,p}(N_e)$ = largo promedio de la cola de trabajos de tipo r que ejecutan con prioridad p, en el dispositivo c del nodo e, cuando hay N_e trabajos en el nodo.

$L_{disco,r}(N_e)$ = largo promedio de la cola de requerimientos a disco, por tipo de trabajo, en el nodo e, cuando hay N_e trabajos en el nodo.

$W_{e,ucp,r,1}(N_e)$ = $S_{e,c,r,1} + \sum_{m=1}^{R_e} (L^r_{e,ucp,m,1}(N_e) - \rho^r_{e,ucp,m,1}(N_e)) * S_{e,ucp,m,1}$

$+ \sum_{m=1}^{R_e} \sum_{j=1}^{2} \rho^r_{e,ucp,m,1}(N_e) * S_{e,ucp,m,j}$

tiempo de respuesta para un trabajo de tipo r que está ejecutando en la UCP, en la máquina e, con prioridad 1, cuando realiza una visita a la UCP.

$W_{e,ucp,r,2}(N_e)$ = $S_{e,ucp,r,2} + \frac{\sum_{m=1}^{R_e} \sum_{j=1}^{2} L^r_{e,ucp,m,j}(N_e) * S_{e,ucp,m,j}}{1 - \sum_{m=1}^{R_e} X^r_{e,ucp,m,1}(N_e) * S_{e,ucp,m,1}}$

tiempo de respuesta para un trabajo de tipo r que está ejecutando en la UCP, en la máquina e, con prioridad 2.

$W_{e,disco,r}(N)$ = $W_{transmision} + S_{disco} * (1 + \sum_{e=1}^{E} \sum_{r=1}^{R_e} \sum_{r=1}^{R_e} L_{disco,r}(N_e - e_r))$

tiempo que se demora una operación a disco en un nodo.

$W_{transmision}$ = tiempo que se demora en transmitir la información a través de la red. Para el caso de una red con carga baja se puede asumir como 0. situación que ocurría en la validación, ver sección 4.

El esquema de modelamiento planteado por Ramamurthy tiene la ventaja de representar adecuadamente el esquema de prioridades de Unix. Pero las desventajas están relacionadas con la labor experimental que se debe realizar para encontrar los parámetros y además, el costo computacional de los cálculos involucrados para evaluar el modelo, aspecto que puede ser verificado en las fórmulas anteriores.

Como una primera labor de investigación, se desarrollo un conjunto de programas para evaluar y validar el modelo de Ramamurthy en ambiente local, lo que permitió ganar experiencia [14], para luego aplicar el estudio a un sistema distribuido con las características

que se especifican en la siguiente sección.

4 Validación del Modelo de Unix en ambiente distribuido

El sistema estudiado, para realizar la validación del modelo de Unix distribuido, estaba compuesto por:

- 5 estaciones de trabajo, SUN 3/50, sin disco local, con sistema operativo Sun 3.
- 1 LAN Ethernet, de 10 Mbits/s.
- 1 servidor con un disco de 200 Mbytes, con sistema operativo Sun3.

En este sistema se ejecutaron un conjunto de programas bien definidos, de manera tal que se pudo cuantificar cada llamada al sistema en forma exacta, utilizando el monitor *trace*. Este conjunto de programas escritos en el lenguaje de programación C se llamaron **prueba1**, **prueba2** y **prueba3**. Prueba1 realizaba la lectura de una matriz de 300x300, ejecutaba un grupo de operaciones y luego, escribía la matriz, en un archivo. Prueba2 era equivalente a prueba1 pero no escribía la matriz. Prueba3 correspondía a la compilación de un programa escrito en C, de 320 líneas.

Se efectuaron pruebas con distintas mezclas de estos programas. Estas pruebas entregaron un conjunto de resultados de tiempo de respuesta, los que se muestran en los gráficos 1, 2 y 3. Para poder comparar los tiempos de respuesta calculados con aquellos medidos, se utilizó el monitor de registro por procesos, conocido como *acctcom*, que es hoy en día una herramienta estándar en Unix [8].

También, se obtuvieron resultados del tiempo de respuesta por tipo de trabajo en el servidor, lo que sirvió para determinar que no hubo un retraso mayor debido a la transmisión. Esta situación se debió a que el grupo de procesos (prueba1, prueba2 y prueba3) no hacia un uso apreciable de la red.

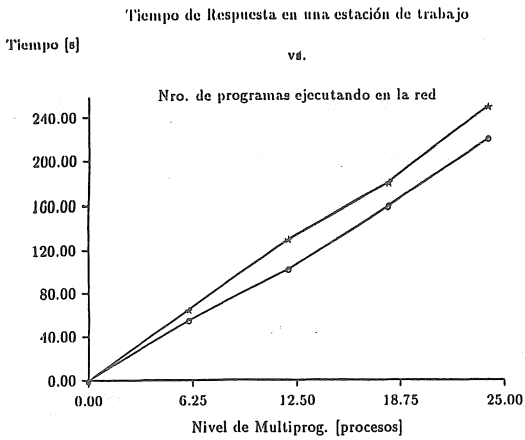
5 Conclusiones

Las conclusiones principales del desarrollo del modelo, aprovechando el modelo desarrollado por Ramamurthy, es que gracias a que la descripción de Unix es bastante cercana a la realidad es posible, sin un costo significativo, modificarlo y adecuarlo a un ambiente distribuido.

La medida de desempeño evaluada, que correspondió al tiempo de respuesta, tiene una precisión adecuada, menor o igual al 10 %, y que permite predecir adecuadamente el comportamiento futuro de un sistema basado en Unix. Es cierto que se requiere de una mayor cantidad de validaciones, pero el hecho de disponer de un modelo como el descrito ya

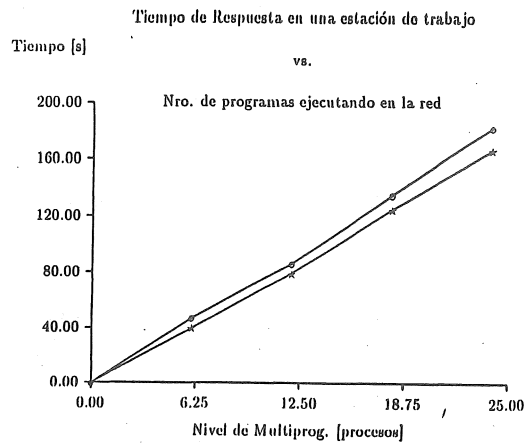
permite avanzar en esta dirección. Un punto de gran interés es poder caracterizar el tipo de proceso que realmente se realiza en una estación de trabajo, en las que en la actualidad ejecutan en ambientes de ventanas y realizan una utilización de recursos excesiva.

Finalmente, el modelo de Unix en ambiente distribuido desarrollado es posible utilizarlo para aumentar su potencialidad, agregando el modelamiento explícito de una red LAN (como Ethernet) [3], u otra, y evaluar una configuración adecuada para un cierto sistema que se desee desarrollar.



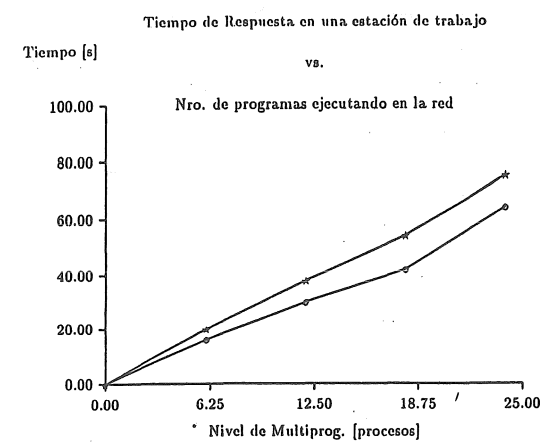
o = tiempo de respuesta calculado para trabajo prueba1
* = tiempo de respuesta real para trabajo prueba1

Gráfico 1. Tiempo de Respuesta en una estación de la red



o = tiempo de respuesta calculado para trabajo prueba2
* = tiempo de respuesta real para trabajo prueba2

Gráfico 2. Tiempo de Respuesta en una estación de la red



o = tiempo de respuesta calculado para trabajo prueba3
* = tiempo de respuesta real para trabajo prueba3

Gráfico 3. Tiempo de Respuesta en una estación de la red

Referencias

[1] M. J. Bach, The Design of the Unix Operating System, Prentice -Hall, Englewood Cliffs, New Jersey, 1986.

[2] K. M. Chandy & D. N. Neuse, Linearizer: A Heuristic Algorithm for Queueing Network Models of Computer Systems, Communications of the ACM, Vol. 25, No. 2, pp. 126-134, February 1982.

[3] J. L. Hammond & P. J. P. ÓReilly, Performance Analysis of L ocal Computer Networks, Addison-Wesley, 1986.

[4] R. Jain, The Art of Computer Systems Performance Analysis, Jo hn Wiley & Sons, 1991.

[5] N. K. Jaiswal, Priority Queues, Academic Press, New York, 196 8.

[6] K. Christian, The Unix Operating System, Wiley-Interscience, 1983.

[7] E. Lazowska et al., Computer Performance Modeling Handbook, Academic Press, New York, 1983.

[8] M. Loukides, Performance Evaluation in Unix, ÓReilly & Asso icattes, 1991.

[9] A. Park and J. C. Becker, Measurements of the Paging Behaviour of Unix, Proceedings of the 1991 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, San Diego, California, pp. 216-217, May 1991.

[10] A. Perez and L. Dowdy, Parameter Interdependencies of File Placement Models in a UNIX System, Proceedings of the 1984 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, Cambridge Massachusetts, pp. 15-26, August 1984.

[11] D. R. Peachey et al., An Experimental Investigation of Scheduling Strategies for Unix, Proceedings of the 1984 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, Cambridge, Massachusetts, pp. 158-166, August 1984.

[12] M. Reiser & S. S. Lavenberg, Mean value analysis of closed multichain queueing networks, Journal of the ACM, Vol. 27, No. 2, pp. 313-322,April 1980.

[13] G. Ramamurthy, An Analytical Model for Unix Systems, AT&T Technical Journal, Vol. 7, No. 5, pp. 86-99, September/October, 1988.

[14] G. Urrea, Evaluación del desempeño de Unix en ambiente local y distribuido, Memoria de Ingeniería Civil en Computación, U. de Chile, 1991.

REASONING WITH EXCEPTIONS FIRST

Marcelino C. Pequeno
Laboratório de Inteligência Artificial
Departamento de Estatística e Computação
Universidade Estadual do Ceará
Avenida Paranjana 1700
E-mail:marcel@ufc.br

ABSTRACT. This paper is concerned with a very basic question: "*What should we infer from a nonmonotonic* knowlegde b*ase?*" We present three principles which we judge essential to correctly answer this question. The second principle *exeception first* is the most important one. It sheds some light upon a very obscure issue in nonmonotonic reasoning: the rising of multiple extensions. It plays a very important role on the formalization of the frame problem. It implies that abnormalities for the frame axiom are calculated step by step instead of globally as all seminal nonmonotonic formalisms do.

KEYWORDS: Common sense reasoning, nonmonotonic logic.

1. INTRODUCTION

This paper is concerned with a very basic question which should be answered before any other consideration is carried out with respect to nonmonotonic reasoning:
"WHAT SHOULD BE THE *DEDUCTIVE* OUTCOME OF A NONMONOTONIC KNOWLEDGE BASE?"

We present three principles which we judge essential to correctly answer this question. The first two are logical principles. They are concerned with the design of a logic modeling common sense reasoning. The third principle is a methodological one. It is concerned with under which (pre)conditions we should use a nonmonotonic logic if we are to have any hope it will satisfactorily model the real life

situation under scrutiny. it is closely related to Carnap's *requirement of total evidence* [2], which is a methodological requirement for the use of inductive logic.

Undoubtedly the second principle which I call *exceptions first* is the most important one. It sheds some light upon a very obscure issue in nonmonotonic reasoning: the rising of multiple extensions, specially in situations where intuition seems to indicate a single alternative. We shall see that the pitfalls pointed out by Hank and McDermott [5] are just to be expected once the *exceptions-first* principle is neglected as it really is by all seminal nonmonotonic logics. *Exceptions first* principle is this sort of things that nobody notices it before, but once it is revealed it looks just too obvious!

The next section describes the first principle. Section 3 presents the second principle. First we enunciate it, then we show that the seminal nomonotonic logics do not comply with it. The final section provides a brief discussion on the issues touched by this paper.

2. PRINCIPLE 1

The general form of a nonmonotonic inference rule is the following:

$$(1) \qquad \frac{P \ ; \ U}{C}$$

Where P represents the premises; U represents the 'exceptions' or undercutters; and C corresponds to the conclusion of the inference.

The nonmonotonic inference rules distinguish from the deductive rules because the undercutters (U) are a sort of *negative* premisses, ie., it is their absence rather than their presence which brings forth the conclusion.

We regard the undercutters (U) as an essential feature of nonmonotonic reasoning. Despite that, undercutters have usually

been negleted in AI. This is certainly true about the seminal papers on the special issue on nonmonotonic reasoning of the *Journal of Artificial Intelligence of 1980.*
Principle 2 next will regulate the interaction between the derivation of undercutters and the application of nonmonotonic rules.

3. PRINCIPLE 2 (EXCEPTIONS FIRST)

Derivations of exceptions to nonmonotonic rules are privileged.

Principle 2 states that considerations on the derivability of an exception to a nonmonotonic rule, should precede considerations on the applicability of the respective rule, ie., if we have reasons to believe in an exception for a rule (even if only defeasible) in a given epistemic situation, then we should believe in it and consequently dismiss the respective rule.

Existing nonmonotonic logics do not comply with this requirement, consequently in some situations the use of a nonmonotonic rule may deter the derivation of its own exception. We believe this fact is entirely counter-intuitive. For example consider the following knowledge situation:

(1) Normally, birds fly unless they are penguins.
(2) Normally, polar birds are penguins.
(3) Penguins do not fly.
(4) Polar birds are birds.

Suppose now that we are concerned about the flyness of polar birds. What should be the outcome of this knowledge situation? Does a polar bird fly or not?

Intuitively, using (2), we conclude that he is a penguin since we have no ground to challenge this inference. Therefore since penguin is an exception to rule (1), rule (1) is

defeated, thus blocking the derivation that he flies. Moreover, by applying rule (3), we conclude that he does not fly after all.

However logics which do not comply with the *exceptions-first* principle (circumscription and default logic for instance) would infer that either a polar bird flies and he is not a penguin, or that he is a penguin and he does not fly (this because they would produce two extensions for this theory, one in which (1) is applied but not (2), and other in which (2) is applied but not (1). Observe that in the first extension we obtain the counter intuitive fact that a rule causes the non-derivation of its own exception).

The exceptions-first principle prescribes exactly that rule (2) should be applied once it leads to an exception (penguin) to rule (1).

3.1. NONMONOTONIC LOGICS DO NOT COMPLY WITH THE EXCEPTIONS-FIRST PRINCIPLE

None of the nonmonotonic logics proposed so far in AI complies with the *exceptions-first* principle. This is true for McCarthy's circumscription [8], Reiter's default logic [17], Moore's auto-epistemic logic [9], Poole's logic [15], etc. We substantiate this claim by presenting a symbolic example which according to the *exceptions-first* principle should lead to a single extension, but when formalized in any of those nonmonotonic logics would lead to multiple extensions or minimal models. Next we present the example and its corresponding formalizations in McCarthy's abnormality formalism, and Reiter's default logic. We chose these two logics to illustrate our point here because we regard them as the most popular nonmonotonic logics among the AI community, however this claim is equally valid to any nonmonotonic logic so far proposed in the literature.

In the following examples we represent a nonmonotonic rule in the form:

Where,

(1) $\dfrac{P\ ;\ U}{C}$

P stands for the positive premises,
U stands for undercutters or exceptions,
C stands for the conclusion.

When the right hand side of the semicolon is empty, it means that there are no undercutters to the rule.

We shall represent rule (1) in the abnormality formalism by the axioms: (2) $P \wedge \sim ab_1 \longrightarrow C$

(3) $U \longrightarrow ab_1$

In default logic, rule (1) will be represent as the multiple semi-normal default:

(4) $\dfrac{P:\ MC\ ,\ M\sim U}{C}$

EXAMPLE I

(5) $\dfrac{B\ ;\ P}{F}$ (6) $\dfrac{PB\ ;}{P}$ (7) $P \longrightarrow \sim F$

(8) $PB \longrightarrow B$ (9) PB

The evidence for *P* in (6) is not challenged, thus according to the *exceptions-first* principle *P* should be inferred and (5) should be consequently defeated. Moreover *~F* (applying (7)) should also be derived.

Observe that our 'penguin example' fits exactly on this structure where *B* stands for *bird*, *P* for *penguin*, *F* for *flies* and *PB* for *polar-bird*.

Now let's see the formalization of this example in McCarthy's abnormality formalism (for details on the abnormality formalism see [8]).

EXAMPLE II

(10) $B \wedge \sim ab_1 \longrightarrow F$ (11) $P \longrightarrow ab_1$ (12) $PB \wedge \sim ab_2 \longrightarrow P$
(13) $P \longrightarrow \sim F$ (14) $PB \longrightarrow B$ (15) PB

This theory has two ab-minimal models one containing {PB, $PB \longrightarrow B$, $P \longrightarrow \sim F$, B, P, $\sim F$, ab_1, $\sim ab_2$}, and other containing {PB, $PB \longrightarrow B$, $P \longrightarrow \sim F$, B, F, $\sim P$, $\sim ab_1$, ab_2}.Thus circumscription only allow us to infer the disjunction $(P \wedge \sim F) \vee (\sim P \wedge F)$. So in this example circumscription has not complied with the *exceptions-first* principle.

Now let's see the formalization of this example in Reiter's default logic (for a more detailed presentation on Reiter's default logic see [17]).

EXAMPLE III

LET T = <W,D>, Where
W = {(18),(19),(20)}
D = {(16),(17)}

(16) $\dfrac{B : MF , M\sim P}{F}$ (17) $\dfrac{PB : MP}{P}$ (18) $P \longrightarrow \sim F$

(19) $PB \longrightarrow B$ (20) PB

This theory has two extensions one containing
{PB, $PB \longrightarrow B$, $P \longrightarrow \sim F$, B, P, $\sim F$}, and other containing
{PB, $PB \longrightarrow B$, $P \longrightarrow \sim F$, B, F, $\sim P$}. Thus in this example default logic does not comply with the *exceptions-first* principle confirming our claim.

4. PRINCIPLE 3

All relevant undercutters for a nonmonotonic rule within a given knowledge situation should be listed.

Principle 3 is a methodological rather than a logical principle. It is concerned with the suitability of a nonmonotonic logic to represent a mundane situation. It guarantees that the logical formalization of a knowledge situation will in fact match reality.

Principle 3 may be regard as the equivalent in nonmonotonic logic for the *requirement of total evidence* in inductive logic. This principle guarantees that a nonmonotonic rule is not applicable in a given knowledge situation unless the total available information in fact supports it.

Nevertheless, this principle is often neglected (even denied) by most of the AI logicians. A common mistake committed by them is to suppose that the use of a nonmonotonic logic would dispense with the necessity of representing exceptions for the nonmonotonic rules altogether.

It is important to point out here that early papers on nonmonotonic logic in AI recognize the importance of taking into account all relevant information available when using a nonmonotonic logic to formalizing a given knowledge situation.

5. CONCLUSIONS AND FURTHER WORK

In this work we presented three principles which we judge essential for the formalization of common sense reasoning. These principles are not complied with by any seminal nonmonotonic logic. We regard this fact as the reason for why these logics fail to formalize some knowledge situations as it has already been pointed out by some authors [5]. In another work [12] we present a nonmonotonic logic DLEF - *Defeasible Logic with Exceptions First* - which is designed according to these principles, and consequently it succeeds in those situations, specially it well formalizes the frame problem. At the moment we are developing a theorem prover for this logic [3,4].

6. REFERENCES

1. Journal of Artificial Intelligence, 13, 1980. Special issue on nonmonotonic reasoning.
2. R, Carnap, *Logical Foundations of Probability,* 2nd ed. Chicago: University of Chicago Press, 1962. First Published in 1951.
3. Godel: Grupo de Estudos em Lógica do Ceará (in portuguese, it means Research Group on Logic from Ceará - Brazil), *Automatic Common Sense Reasoning.* Current CNPq project grant no. 500044/90-2.
4. Godel: Grupo de Estudos em Lógica do Ceará (in portuguese, it means Research Group on Logic from Ceará - Brazil), *Automatic Common Sense Reasoning.* Current CNPq project grant no. 500044/90-2.
5. S. Hanks, & D. McDermott, *Nonmonotonic Logic and Temporal Projection,* Artificial Intelligence 33, 1987.
6. R. Loui, *Response to Hanks and McDermott: Temporal Evolution of Beliefs and Beliefs aboutTemporal Evolution,* Cognitive Science, 11 1987
7. J. McCarthy, *Applications of Circumscription to Formalizing Common Sense Knowledge,* Proceedings of the American Association for Artificial Intelligence, 28, 1986.
8. J. McCarthy, *Applications of Circumscription to Formalizing Common Sense Knowledge,* Artificial Intelligence 28, 1986.
9. R. C. Moore, *Semantical Considerations on Nonmonotonic Logic.* Proceedings of the Eighth IJCAI, Karlsruhe, West Germany, August 1983.
10. M. C. Pequeno, *Normally Canaries Fly,* In: Anais of the VI Brazilan Symposium on Artificial Intelligence, Rio de Janeiro, November, 1989.
11. M. C. Pequeno, *Three Principles for the Formalization of Nonmonotonic Reasoning.* In: Anais of the VIII Brazilan Symposium on Artificial Intelligence, Brasília, November, 1991.
12. M. C. Pequeno, *Nonmonotonic Reasoning with Exceptions First,* submitted PhD thesis, Department of Computing, Imperial College, 1992.
13. T.H.C. Pequeno, *Inconsistent Default Logic,* Technical Reporter, Department of Computing, Imperial College, 1989.
14. T. H. C. Pequeno, and A. Buchsbaum, *The Logic of Epistemic Inconsistency,* Proceedings of the Second International Conference on Principles of Knowledge Representation and reasoning, Cambrigde, Masschusetts, April 22-25, 1991.
15. D.L. Poole, *Default and Conjectures: Hypothetical Reasoning for Explanation and Prediction,* Research Report CS-87-54, University of Waterloo, Ontario, 1987.
16. R. Reiter, & G. Criscuolo, *Some Representational Issues in Default Reasoning.* Tech Rep. No. 80-7, Department of Computer Science, University of Bristish Columbia, Vancouver, 1980.
17. R. Reiter, *A Logic for Default Reasoning,* Artificial Intelligence 13, 1980.

PRESENTATION OF AMARGO: AN HIERARCHICAL VLSI PHYSICAL LAYOUT SYNTHESIS SYSTEM

CARLOS EDUARDO PEREIRA

INSTITUT FUR REGELUNGSRECHNIK UND
PROZESSAUTOMATISIERUN
UNIV. STUTTGART
PFAFFENWALDRING, 47
STUTTGART 80 G-7000
GERMANY

DANTE AUGUSTO COUTO BARONE

INSTITUTO DE INFORMÁTICA
UNIV. FEDERAL DO RIO GRANDE DO SUL
CAIXA POSTAL 15064 - CEP 91501-970
PORTO ALEGRE - RS - BRASIL
PHONE: (051) 336-8399 FAX: (051) 336-5576
E-MAIL ADDRESS: BARONE@ INF.UFRGS.BR

ABSTRACT

This work describes AMARGO, an hierarquical environment for general cell VLSI routing, which consists of a number of procedures as a global router, a router for power and ground lines and a channel rounter AMARGO belongs to the Cp2 project: Parallel Processing Contributions: Advances in Microelectronics and in Computer Architecture under development at the Instituto de Informática of the Federal University of Rio Grande do Sul - Brazil

KEY-WORDS:

general cell - VLSI routing - channel routing - Cp2

The Cp2 project is funded by CNPq (Brazilian Council For Research).

I. INTRODUCTION

This paper will describes AMARGO (in portuguese, AMbiente para Automação do Roteamento Geral de CircuiOs), an interactive placement and a automatic routing tools. These tools belong to the layout synthesis abstraction level. The modularisation's strategy of this tool, the proposed algorithms and the obtained results in each software module will be briefly presented.

AMARGO can be classified as a "general-cell" layout synthesis system to be used in the final assembly of the circuit into the chip. The utilisation of this tool is fundamental in the physical layout phase of complex integrated circuits.

Some complex I.C. developed at the Microelectronics Group (GME) of the Federal University of Rio Grande do Sul as RISC microprocessor (PCIR) and a graphics controller (CVTX), had the layout synthesis phase restricted to the mask definition of some main blocks without completing the interconnections between then due to the absence of appropriate tools.

Layout systems that address the same problem as AMARGO have been manufacturated by great software vendors, at costs higher than US$ 50.000. In spite of this extreme extensive cost, many drawbacks can be imputable to these commercial tools. This increases the complexity in the development of a layout physical synthesis system as AMARGO.

II DESCRIPTION OF THE AMARGO SYSTEM

To use the AMARGO system the I.C. designer must provide the synthesis of the functional blocks of the circuit. These blocks can be obtained automatically, by the activation of module generators and choice of the blocks parameters, or since the retrieval of appropriate cells from a cell's library.

Once the layout of all modules of a specif circuit under design is completed, the interconnections among them must be established. This interconnections definition may be done interactively, by the appointment of the connectors which belong to

each interconnection (net) or by a description of a text-file as the described below in table 1.

$ NET
< name-net >
< name-conector> <instance which the connector belongs to >
.
.
.
< name-conector> <instance which the connector belongs to >

Table 1: Text-file of a Net

In fig. 1 it is showed a possible configuration to the system's input. In this case four functional blocks must be interconnected.

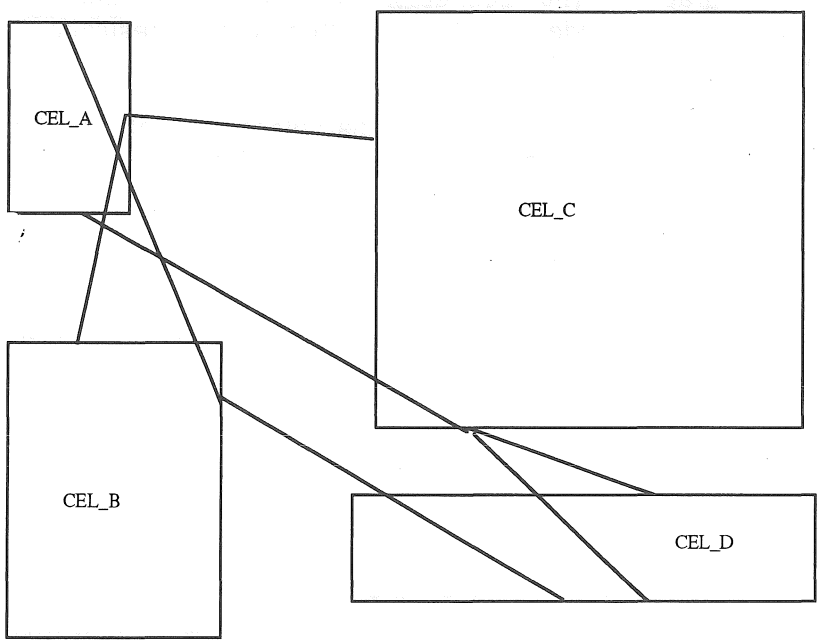


Fig. 1 - Possible configuration to AMARGO's input

Once completed the relative placement (the position of the blocks is not fixed) added to the definition of the nets of the circuit, the AMARGO system's activation will proceed the following tasks:

1) initial placement adjustment;
2) global routing;
3) power lines routing;
4) channels ordering;
5) channel routing.

All these tasks are performed automatically. Meanwhile the designer can influence by the bias of the algorithms parametrization. To facilitate the results interpretation task, reports containing many relevant statistics are generated in each phase. These reports can be re-used by systems which allow decisions re-evaluation providing then backtracking.

To allow the global routing, it must be obtained a placement graph. The connectors of the blocks that will be interconnected are added to the placement graph, resulting in the so-called complete graph.

If it is disponible the initial placement of fig. 2-a, its corresponding global routing is showed in fig. 2-b.

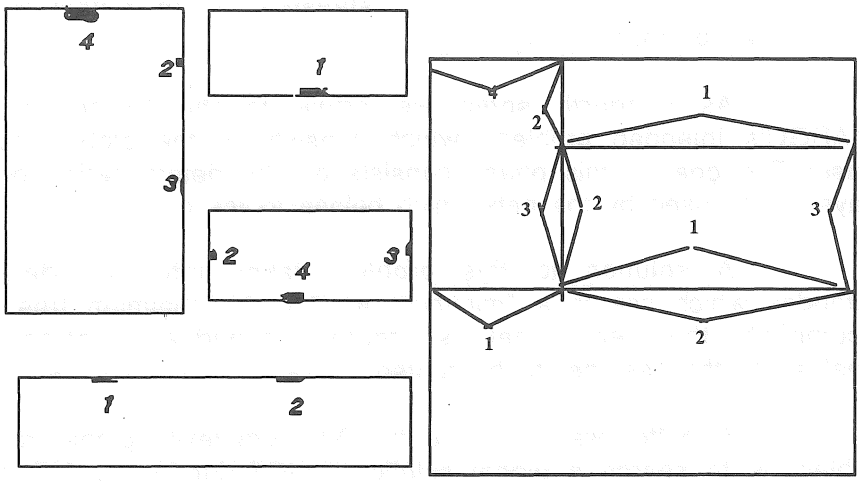


Fig. 2 - Example showing an initial placement and the corresponding complete graph.

The AMARGO system solves the generical routing problem taking into consideration a divide-to-conquer methodology This is done to keep in reasonably leves the compleixty of the envolved task.

Initially, all routing areas are identified. This is accomplished to obtain all free spaces to the routing task. The solution to this problem employs the corner-stitching data structure technique. In this sense all rectangles are mapped, even the empty spaces.

Secondly, the transparecny over-the-cell channels are established to determine which net will take each transparency of the modules. This claims to minimize the routing area. A solution to this problem consists on the determination of the minimum bounding box (MBB) which contains all net's connectors. To each transparency T of a block B, the following tasks must be performed:

- prepare a list of nets which MBB includes the block into consideration;
- choice of a net, taking into consideration factors as priority and net of the largest channel's length (trying to avoid it)

As a third step, an adjustment of the placement is performed. This allows to verify if there is sufficient space between the blocks to constitute routing areas. A solution to this

problem evolves the utilisation of statistics. More details about this task can be found in [1].

As a fourth sptep we arrive to the Kernel of the AMARGO's intended problem, which consists of the global routing phase. The goal in this phase consists on the determination of the "ways" to be used by the nets which belong to the circuit.

A solution to this problem takes into consideration methodos which search a "minimum way" or a "minimum tree". To accomplish this, each net is treated individualy, introducing penalties to that last net to be routed.

A better way to solve the NP complexity global routing problem is to search a global solution considering not a "minimum tree". but a "minimum forest".

In the minimum forest approach, a global solution is optimized presenting the following characteristics:

- independence of the nets ordering
- simultaneity of the development of all trees.

The minimum forest solution employs extensively graphs to model the routing problem, not only preparing placement graphs as conectors graphs .

As the corner-stitching is employed the topological and geometrical models are joined together. In the solution, each vertex corresponds to an intersection between rectangles and each edge corresponds to a congested space.

The proposed and original minimum forest algorithm is presented in table 2.

A) Starting from an empty graph;
B) the minimum weight egde to be introduced in the graph is chosen. This edge cannot produce cycles with the other net's edges;
C) The B step is repeated until all connectors of the nets be connected.

Table 2 - Proposed Minimum Forest Algorithm

As a fifth step the power lines routing (GND/VDD) is produced aiming to minimize the potential drops and metalic migration in the routing and dimensioning of the VDD and GND tracks task. To accomplish this, the power lines global routing is performed prior to the other nets. The determination of each line's width depends on statics and dynamics factors. Finally, a detailed routing is performed

As a sixty step the ordering of the channels is accomplished. Thes step has as goal the establishmet of a chronological order for the channels routing avoiding interferences among them.

As a seventh step the complex channel routing phase is performed. In the case of the AMARGO system, a channel router tool yet developped at GME, named SCHAROP was employed. For more details the [2] reference is suggested.

III. CONCLUSIONS

The present situation of the AMARGO project is the following:

- Graphics interface: implemented and tested.

- Data structure interface: specified and 80% implemented. This interface was developed having in mind a future data base management system integration.

- All main AMARGO's routing steps had their algorithms entirely specified and many of the them completely implemented.

The hardware and software platform used In the project consists of a IBM - PC microcomputer.

As the system under design is extremely complex, the implemented platform is far from beeing the ideal. As a sequence of this project, it is intended to employ more sophisticated environments, as SUN/UNIX workstations.

REFERENCES

[1] PEREIRA, C. E. ; BARONE, D. A. C. SCHAROP - A Detailed Channel Router. In: IV Brazilian Microelectronics Society Congress, Porto Alegre, july 12-14, 1989. Proceedings. p. 849-859

[2] PEREIRA, C. E.. AMARGO: An hieranchal environment for integrated circuits assembly. M. Sc. dissertation. Pós-Graduação em Ciência da Computação, Porto Alegre, 1990.

EXPLANATION CONSTRAINTS IN A MULTI-AGENT ENVIRONMENT

Authors: PIQUEMAL-BALUARD C., GLIZE P.
Address: IRIT - Université Paul Sabatier
118 route de Narbonne
31062- Toulouse cedex
France
Email: Piquemal@irit.fr
Phone: 61556611 poste 7253
Fax: 61556258

ABSTRACT:

The explicit separation of the knowledge and the system that uses them (the inference's engine) offers to expert systems dialog abilities forbidden to programs built on traditional data processing technics.

This capacity of dialog and self-explanation must be used and developed because it represents one of the main advantages of this kind of system. Futhermore the qualities of user friendship involved in the installation of a mechanism of explanation, this allows particularly the systems to be validated. The justification of the results increases in a large way the confidence accorded by user to the solution found.

We present a study of the explanation in the context of a multi-agent system with one unique explanatory module. Because of the constraints of this kind of software, the explanation cannot be based on the addition of specific explanatory knowledge on each form that can be, but only on syntactic analysis of the knowledge and user model. We analyse the consequences of this approach and we give examples on each part in the medical field.

Keywords: explanation, user models, multi-agent systems, man-machine communication, knowledge representation.

I- INTRODUCTION

The addition of an explanatory mechanism into a Knowledge Based System (KBS) is necessary to provide an intelligent description of the expert knowledge and of the reasoning process done with it. Its integration is facilitated by the architecture of KBS where the explicit separation between knowledge and inference engine gives dialog abilities unattainable from classical programs [14]. An explanatory mechanism increases "user friendship" and confidence in the proposed solution. The first works about explanation [6], [16], [17] have provided to the user the inferences leading to a given result. Unfortunately, the lack of discrimination between the message parts is prejudical to the clearness and of inability to explain the terminology.

The following works done on explanation have given much different results. The first reason for that is the adaptation of the explanatory mechanism to the structure of the KBS (its knowledge representation, its inference engine architecture characteristics, and even the expertise field). Finally, it's very difficult to formalise the explanation process, which must answer to the following points [1]:

- To whom is the information addressed?
- What kind of information?
- The granularity of the information?
- In what form of presentation?
- When can it be available?

II- EXPLANATORY MESSAGE ELABORATION

If the elements of an explanatory message are extracted from the knowledge, they must be chosen from some criteria. The sending informations are selected according to their relevance (concepts are more essential than others) and the user nature.

2.1 The knowledge classification by addition of explanatory informations

In this method, the role of each knowledge element is defined by the expert in adding its justification into the knowledge base. The notion of structural knowledge developed by NEOMYCIN consists in the classification

of the knowledge domain in many linked groups. A specific justification method is associated to each group [6].

ADELE [13] put in two separate modules the surface knowledge and the deep knowledge. The deep knowledge module contains theories and resolution knowledge that are specific to the domains. They allow in particular the description of links between objects, between objects and object's classes, leading to the construction of semantic networks.

These "ad hoc" informations are of multiple kinds. They can be written in a message form to facilitate their using, or in belonging to a class.

But, this method set the problems of the memory space and of its updates when the expert knowledge base is modified. Then, explanations in the multi-agent system SYNERGIC [15] are build without adding "explanatory" informations. We analyse further the associated constraints and limits.

2.2 The user modelling by autopresentation

The explanatory message is sent to a particular user that has a given competence when the session starts. His competence level and his expectations are important criteria to build a pertinent answer.

The easier way (and the most frequently used) to know that is to begin a session with a user autopresentation. It may generally consist in asking for his competence domains and the desired degree of explanation development. Inside the Knowledge base, each concept (rules and objects) is evaluated by the expert. This value represents their complexity and understanding degree. The explanatory message contains only rules or objects having a value situated between the domain competence and the explanation degree [17].

BOUZEGHOUD [3] constructs the explanation according to the initial competence degree of the user. For an expert, it consists only in editing the rules without comments, to improve the knowledge if necessary. A designer concerned in the application modelling, receives the closely related rules with comments or graphics. Whereas all the underlying knowledge is edited for a student.

2.3 The method of dynamic user modelling

However, we are never sure of the user judgement: he could underestimate himself. In addition, the same user cannot understand all the concepts of a knowledge base with the same degree. A question to the user about each of them is unrealistic.

For these reasons we have chosen to use a method of dynamic modelling in SYNERGIC. The explanatory mechanism implicitly judges the user competence during the dialog. It has been already employed in another software [4].

III- THE EXPLANATION IN THE MULTI-AGENT SYSTEM SYNERGIC

3.1 Context presentation

SYNERGIC is a multi-agent system where the solution of the problem is the result of a team co-operation. Each team member is an agent, classical procedures considered by SYNERGIC as a black box, or a knowledge module that can be explained.

Each knowledge module is hierarchically ordered in levels. Each level has the metaknowledge of the lower level. The upper level defines the general strategy to solve domain problems (the surface knowledge): it is the patient interview and the clinical examination in medicine. The lowest level contains the detailed informations necessary to solve the high level tasks (deep knowledge). It is the symptom's study realised by the medical expert to find the potential diagnosis [15].

3.2 The explanatory informations about the domain

An expert system shell ought to explain the reasoning process in many diverse domains. In the same way, the multi-agent system shell SYNERGIC could possess many expertise's fields in a same application. Thus, an explanatory mechanism must be able to work in a domain in an independent manner.

Usually, a specific explanatory module is associated with each knowledge module. The explanation built with this method is thus very closed to the domain, but many problems occur. Essentially, the memory space and the update, as we have seen in 2.1. On top of that, when the explanatory knowledge is included in the knowledge base, it is difficult to do the difference between the solving knowledge and the explaining knowledge. Agreing the maxim "all erudite is not a good pedagogue" [13], we think that a good field expert is not necessarily able to create the explanatory knowledge.

Thus, we have chosen in SYNERGIC a method that allows a real domain independent explanation. Because it cannot use the real meaning of the employed terminology, it finds them with a syntactic analysis (parsing) of the knowledge base, to create an explanatory message in natural language.

The answer elaboration imposes to select the informations to send. The main criterion of this pruning is the discovery of the event role inside the module and during the solving process.

An event role is understood with its internal characteristics (or structural informations) and its links with the other events of the domain (or contextual informations).

3.3 The structural informations to classify the knowledge

The first explanatory information is derived from the event decomposition in each of its elementary components: object, attribute, link, value. The citations of **palpation, abdomen, causes** and **pain** are the beginning to explain the event **palpation of abdomen causes pain. Temperature** is a property of the object **patient** for the event **the temperature of patient> 37.2** while **37.2** is a possible value for the entity **temperature of patient.** These indications are helps to define the explanation degree associated to an object.

The following indications are derived from the rules containing the event. An event only situated in premises is an external information obtained with a user dialog. Thus he knows the event existence and its value, and it is not necessary to develop it in the explanatory message.

The last indication is the level of the event. The hierarchical structure of knowledge bases into levels allows to solve the problem of strategic knowledge dilution.

3.4 The contextual informations to classify the knowledge

The event role is completed by an analysis of its links with the remaining knowledge. But the causal links (production rules) can have many meanings: implication, rewriting, procedure call [3].

A directed syntactical analysis is able to furnish these meanings without adding explanatory informations. For instance, common entities between different condition sets inferring an event, is spelled necessary conditions (but not sufficient).

The influence degree of an event on another can thus be evaluated. It is meaningfulless when events are related to the same entity: each value role or attribute's role of an object can be discovered with this method.

3.5 The method of dynamic user modelling

Knowledge about user (interest subjects, domain competence,...) are memorised inside the user model. Even if the explanation goal is not teaching, he can improve his knowledge. Thus, a better understanding of concepts, and the discovery of another must be echoed in the user model: it is the method of dynamic user modelling.

The explanatory mechanism evaluates implicitly the user, without initial autopresentation. This method increases the "user friendliness" and it is a gain of time. The main pieces of information are the questions of the user. An event has a privileged treatment when it is the subject of many questions.

The **dialog trace** is thus necessary for the dynamic modelling. The dialog trace allows also to repeat the same explanations and to detect implicit questions in the user discourse. An increasing interest or a lack of understanding of the explanation could be the underlying reason. The study of the user questions is a good help to find his subjects of interest; and the study of the answers improves their relevance.

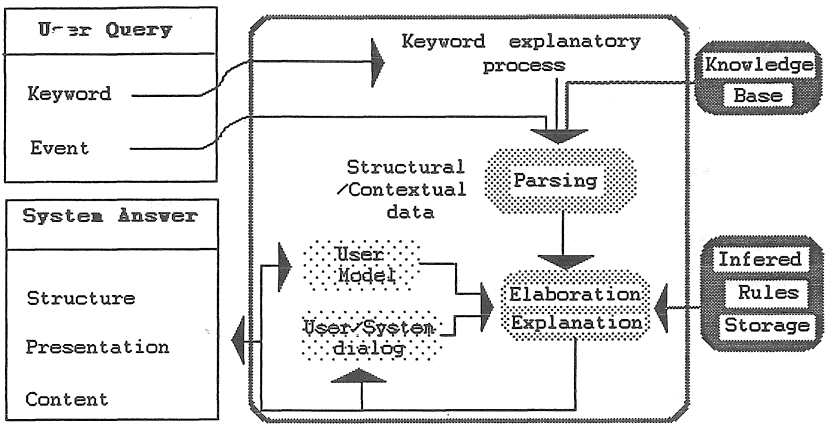
3.6 The analysis granularity

The study of all the links between the events leads usually to a great amount of information and it is time consuming. To reduce this amount of structural and contextual informations, the explanatory mechanism drives the research about an object or an entity. The importance of the event and its links with the other entities of the domain are the criteria used to limit the search.

IV- THE EXPLANATION PROCESS OF SYNERGIC

This process is based on question/answer principle. A question is composed of a keyword and the explanation subject (the event). The user can set three kinds of queries. One is associated to an explanation about the knowledge (WHAT), the others are used to understand the process reasoning during a session (WHY) or at the end (HOW). Each of them owns a specific standard answer, adaptable at user and knowledge.

Fig.1 Interactions between the elements of the explanatory mechanism



4.1 Example of the explanatory method associated to the keyword WHY

In order to show the explanation process, we develop the answer elaboration to the question **WHY an event?**. In our case, the user model is empty because this question starts the dialog.

The event is located in first position in the reasoning process. The answer is added with informations about its real role in the knowledge. Their location in the answer and their granularity depends on the amount of the previously edited informations.

4.1.1 Location of the event in the reasoning process

The short term explanation is deduced from the last fired rule having this event in its premise. It is found in the trace and edited in a simplified natural language form.

The long term explanation consists in finding the context of the inference of this rule. The pruning is made with the two previous criteria: the user model and the place of the event in the knowledge. This pruning allows to catch the user eye on the most significant results to obtain.

4.1.2 The event meaning

The previous answer is completed with short definitions of the event and the entities contained in the justification. An event not deduced by the knowledge (an initial fact or a user fact) is situated in the domain. The

effects of its valuation on the knowledge base are edited (except the last fired rule).

In order to better understand justification, the premise of the rules not fired and containing it are written. The consequence of its determination on the reasoning process are then listed.

4.2 Example of explanation in medecine

The example of a question WHY on TELEMAC is analyzed. TELEMAC [15] is an application of SYNERGIC in the medical domain developed in cooperation with the Centre Hospitalier of Toulouse. The knowledge base currently active is the "thoracic pain" which has 250 rules. Among the 19 possible diagnosis, it tries to find the plausible pathologies for a patient having an acute thoracic pain.

At this stage, the inference engine has no inferable rule, and the man-machine interface asks the user about the pain topography to continue the reasoning process. But the user wants to know the goal of this question before his answer. Thus, his first need of explanation in this session is: **WHY the topography of pain?**

This sentence begining with the keyword WHY leads to the loading of the explanation module. Without any other dialog the system writes:

The search of topography of pain begins the study of features of pain. It should be followed by the determination of irradiation, starting and type.

It allows the deduction of objects to probable or uncertain. An object is member of the set: (colonopathie, acute-appendicitis,..).

In long term, it leads to the initialization of probability of diagnosis allowing the plausible deduction of diagnosis of abdominal = realised.

This result is composed of three parts, precisely ordered. The first part concerns location, in the reasoning process, of the explanation subject (the event) for the landmark of the user. The second part defines the event role independently of the session (the principle of explanation-based example). The last part gives a more general view of the reasoning: the final result and perhaps further questions.

After the message editing the user can restart the dialog. The answer and the questions modify dynamically the user model.

4.3 First message part

It refers to the last fired rule in backward chaining. It locates where the inference engine stops.

```
/* RULE 17 */    If the topography of pain
                 and the irradiation of pain
                 and the  starting of pain
                 and the typeof pain
                 then the features of pain
```

Each premise is sent to define the immediate context of the event. It explain why the system is stopped, and the others informations missing to fire the rule.

Because all the premise concern the same object **pain**, they are written in a shorten form: only the attributes are given. The verb **begins** is employed because the event starts the rule. The terminology used by the explanatory mechanism is therefore particularly chosen to increase the answer precision. The mechanism possesses also a set of predefined sentences to adjust the responses to the knowledge and situations.

4.4 Second message part

In order to locate the event in the knowledge, its function and its impact on the remaining knowledge are presented. A syntactic analysis centered on **topography of pain** allows the explanatory mechanism to find that it is only able to deduce **diagnosis of X = probable or uncertain**. X is a member of the object set having **probability** and **diagnosis** as attributes. These objects are very significant for this reason and justify their presence in the explanatory message.

Nevertheless, the explanation module is unable to discover that each X is one the 19 possible diagnosis. The enumeration of the set overcomes this problem.

4.5 Third message part

The explanation ends with the fact justifying the existence of **topography of pain**. It was selected from the set of rules leading to the conclusion. For instance, all the entities member of this set (as **features of pain**) used to order the resolution are excluded. The conclusion of the last fired rule in backward chaining is also excluded. The complete syntactical analysis of the related rules leads to the great relevant entity **probability of diagnosis**.

The syntactic analysis reveals also that this entity is valuated and has an initial value, obtained at the end of the path starting with the event. For this reason, the explanatory mechanism employs the term initialization.

V- CONCLUSION

The integration of an explanation mechanism in a multi-agent system comme up against the difficulty of knowledge diversity. In effect, the explanation is usually domain dependent. More, an expert uses a specific domain terminology. These reasons leads to think that a knowledge base must be **autosufficient for explain**, allowing a really domain independent explanation mechanism. Our method and its results largely justify this principle.

During the knowledge acquisition process, the expert generally includes justifications. During the transcription phase of the interviews in formalized knowledge by a cognitician, these justifications are generally suppressed because they are not useful for problem solving. We think that the shared knowledge notion in a multi-agent system, allows us to retrieve some of these explanatory knowledge in others modules. When we lack of justification in the currently module, another related module would be able to give supplementary informations. It is our future work for the explanation in a multi-agent system.

BIBLIOGRAPHY:

[1] BOURI M,DIENG R, KASSEL G, SAFAR B: Systèmes à base de connaissances et Explication PRC-GDRIA, 3ième journées nationales, 5-7 Mars 1990

[2] BOURI M,DIENG R, KASSEL G, SAFAR B: Vers des SE plus explicatifsPRC-GDRIA, 3ième journées nationales, 5-7 Mars 1990

[3] BOUZEGHOUD M, METAIS E: La justification des résultats et l'explication du raisonnement dans le système expert SECSI. CIAM 86

[4] BREZILLON P: Intervention de l'utilisateur dans les explication Deuxièmes journées explication du PRC-GDR-IA du CNRS Sophia-Antipolis 17-19 juin 1992

[5] CHOURAQUI E: Un modèle formel de caractérisation de l'évolution des connaissances. Sixième conférence Canadienne sur L'Intelligence Artificielle. Montréal, 1986

[6] CLANCEY W. J, HASLING D. W., RENNELS G.: Strategic explanations for a diagnostic consultation system.Int. J. Man-Machine studies 20 1984

[7] DAVID J. M, KRIVINE J. P: Explaining Raisoning from Knowledge Level Models.ECAI 1990

[8] DIENG R: Explications à partir d'arbres de raisonnement étiquetés par des relations. Journées Explications du PRC IA,Mars 1989

[9] ERIKSON R, JOHANSSON A. L.: Neat explanation of proof trees IJCAI 85

[10] FININ T, DRAGER D.: GUMS: A general User Modelling System Sixième conférence Canadienne sur L'Intelligence Artificielle. Montréal, 1986

[11] GILBERT N, BUCKLAND S, FROHLICH D, JIROTKA M, LUFF P: Providing Advice through dialog. ACAI 1990

[12] JIMENEZ DOMINGUEZ C: Le problème des connaissances implicites pour l'explication Colloque IA Caen 14, 18 Septembre 1987

[13] KASSEL G: Système Expert de seconde génération et explication du raisonnement. Avignon 1989

[14] LEMAIRE B,SAFAR B.: Some necessary features for explanation planning architectures: study and proposal, in proceedings of AAAI'91 Workshop on Explanations. Anaheim, USA, July 1991

[15] PUYET P, CARPUAT B, GLIZE P, PUJOS M, MAGUES J.P: TELEMAC: a multi-agent system for aided-decision and diagnosis approach. AAAI Springs Symposium of Artificial Intelligence in Medecine, March 1992

[16] SWARTOUT W. R: XPLAIN: a system for creating and Explaining Expert Consulting Programs.AI 21(1983), 285-325

[17] WALLIS J. W, SHORTLIFE E. H.: Customized Explanations Using Causal Knowledge, Rule Based Expert Systems. Addison-Wesley, 1984

O Desenvolvimento de um Sistema de Diretório

Luci Pirmez

Núcleo de Computação Eletrônica - UFRJ,

Caixa Postal: 2324, tel:(021) 598-3165,

Rio de Janeiro, RJ, Brasil

E-mail :NCD10122@UFRJ.BITNET, FAX:(021) 270-8554.

30 junho 1992

Abstract

O objetivo deste artigo é apresentar a experiência adquirida no desenvolvimento de um Sistema de Diretório dentro do projeto Inter-X. Inicialmente são descritos os conceitos básicos do Sistema de Diretório e introduzidos os modelos funcional, organizacional e o de informação e em seguida seu funcionamento. Por fim, é descrito o ambiente, a estrutura e o processo de validação da implementação.

Palavra Chave : Redes de Computadores, Protocolos, Modelo OSI, Diretório.

1 Introdução

A sociedade moderna vem procurando ao longo dos anos, desde o surgimento do telefone, aumentar cada vez mais sua capacidade de comunicação em escala internacional.

A expectativa projetada para o final do século é o surgimento de novos serviços, aliado a uma maior integração dos serviços atualmente oferecidos, com a utilização intensa de novas tecnologias digitais em redes.

Para tornar possível o fornecimento de serviços de Diretório para seus usuários de modo a permitir o acesso as informações sobre objetos armazenados em diferentes sistemas abertos, o CCITT deliberou uma série de recomendações que define um conjunto de procedimentos padrões conhecidos como Recomendação X.500 e que são encontradas no "Blue Book"

1.1 O Desenvolvimento do Sistema de Diretório no NCE/UFRJ

A especificação de um Sistema de Diretório (DS) e futuramente sua implementação fazem parte de um projeto denominado INTER-X. Este projeto tem como objetivo principal gerar um Sistema de Manipulação de Mensagem (MHS) avançado e distribuído acoplado a um Sistema de Diretório padronizado que facilite a interação entre usuários na identificação e endereçamento dos usuários além do gerenciamento de outras funções do sistema como, roteamento de mensgem e armazenamento de lista de distribuição.

A implementação dos serviços a serem oferecidos pelo INTER-X seguem a tendência internacional de basear os desenvolvimentos de software segundo o Modelo de Referência OSI (Open Systems Interconnection) da ISO. Uma outra tendência observada é a convivência do TCP/IP com o modelo OSI. Neste projeto a RFC 1006 será utilizada para possibilitar o acoplamento do transporte OSI com o transporte (TCP) da família TCP/IP permitindo que as aplicações X.500 e X.400 rodem sobre uma rede TCP/IP.

A rede TCP/IP será utilizada como meio de interconexão entre os usuários localizados em diferentes sistemas abertos. O hardware e o software necessários para permitir a interconexão serão adquiridos diretamente do fabricante. Para a camada de transporte e sessão da ISO serão utilizados aqueles desenvolvidos e validados pelo nosso grupo de redes no ambiente VAX/VMS da digital e portados para o ambiente SUN..

2 O Diretório

Diretório é um depósito de informações sobre objetos e fornece Serviços de Diretório para seus usuários de modo a permitir o acesso as informações. As informações armazenadas no Diretório são denominadas de Base de Informação de Diretório (DIB).

O Diretório fornece um conjunto de facilidades de acesso para seus usuários, denominadas Serviços Abstratos do Diretório. A utilização dos Serviços de Diretório requer que os usuários e os componentes funcionais do Diretório cooperem entre si. Tal cooperação também é necessária entre aplicações em diferentes sistemas abertos, e para tal necessitam utilizar protocolos padronizados.

O Diretório foi projetado para atender a várias aplicações. A natureza da aplicação suportada determinará quais objetos serão obtidos do Diretório, quais usuários acessam a informação e os tipos de acesso permitidos.

2.1 O Diretório e seus Usuários

O usuário do Diretório é a entidade (processo de aplicação) ou pessoa que acessa o Diretório com o propósito de obter Serviços de Diretório (DS). Cada usuário é representado no acesso a um Diretório por um Agente de Diretório do Usuário (DUA). Mais precisamente, é o DUA que realmente acessa o Diretório a fim de obter serviços do interesse de um usuário. Cada DUA representa um usuário do Diretório e é considerado um processo de aplicação.

2.2 O Modelo Funcional e Organizacional do Diretório

O Diretório é composto de um ou mais processos de aplicação denominados Agentes de Sistema de Diretório (DSA). Cada DSA fornece zero ou mais pontos de acesso. Quando o Diretório é composto por mais de um DSA ele é dito ser distribuído.

O papel do DSA é prover aos seus DUA's e/ou outros DSAs o acesso à DIB. Um DSA pode usar informações armazenadas em sua base de dados local ou interagir com outros DAS's para executar operaçãoes. Base de dados locais são totalmente dependentes da implementação.

O conjunto de um ou mais DSAs e zero ou mais DUAs gerenciados por uma única organização formam um Domínio de Gerenciamento de Diretório (DMD). Um DMD pode ser um Domínio de Gerenciamento Administrativo de Diretório (ADDMD) ou Domínio

de Gerenciamento Privado de Diretório (PRDMD), dependendo de estar sendo operada por uma organização privada ou por uma organização pública de telecomunicação, respectivamente.

2.3 Modos de Operação do Modelo do Diretório

O DUA interage com o Diretório através da comunicação com um ou mais DSAs. Ele pode interagir com vários DSAs para solicitar a execução de operações. Pode não ser sempre possível interagir diretamente com o DSA onde o pedido será executado, isto é, retornar informações do Diretório.

Um DSA é responsável pela execução de operações solicitados pelos DUAs e pela obtenção de informação quando este DSA não o tem. Para obter informações do interesse de um DUA, um DSA terá que interagir com outros DSAs. Foram identificados três modos de como as operações de um DUA podem ser manipulados:

- **Referral -** Um resultado é retornado por um DSA quando este não pode executar a operação, e este resultado identifica um ou mais DSA capazes de executar esta operação.
- **Encadeamento (Chaining) -** Modo de interação usado opcionalmente por um DSA quando este não pode executar a operação. O DSA encadeia a operação enviando a mesma operação para outro DSA e retransmitindo o resultado para o originador do pedido.
- **Multicasting -** Modo de interação usado opcionalmente por um DSA quando este não pode executar a operação. O DSA envia a operação para vários outros DSAs (em série ou em paralelo) e retorna o resultado apropriado ao originador.

2.4 Base de Informação do Diretório (DIB)

A DIB é composta de informações sobre objetos. É composta de entradas, cada uma contendo um conjunto de informações sobre um objeto. Cada entrada por sua vez, é formada por um conjunto de atributos onde cada atributo contém seu tipo e um ou mais valores. A recomendação X.520 define os diferentes tipos de atributos que são úteis para aplicações que utilizam Diretório, como por exemplo os atributos geográficos: nome do pais, nome da localidade, nome do estado.

Cada tipo de atributo presente em uma determinada entrada é dependente da classe do objeto a qual descreve a entrada. A recomendação X.521 define as diferentes classes de objetos úteis para aplicações que utilizam o Diretório, como por exemplo: pessoa residencial, pessoa organizacional, processo de aplicação.

As entradas da DIB estão organizadas em forma de árvore, a Árvore de Informação de Diretório (DIT) onde os vértices representam as entradas.As entradas da DIB são identificadas por nomes. Cada entrada tem um nome distinto (DN), e este nome identifica a entrada unicamente e sem ambiguidade. Esta propriedade de nome distinto é derivada da estrutura em árvore da informação. O nome distinto de uma entrada é composto pelo nome distinto da sua entrada superior e pelos valores distintos da entrada. Os objetos da entrada são distinguíveis através de seus nomes relativos (RDN).

3 Análise dos Serviços do Sistema Diretório

3.1 Estrutura da Rede de Telecomunicações

Dada a grande heterogeneidade de arquitetura dos sistemas de comunicação, é importante ter em mente o fator compatibilidade na interconexão de rede. O modelo funcional dos DS define dois agentes responsáveis pelo fornecimento de serviços aos usuários do sistema: o DUA e o DSA. A estrutura interna destes agentes não está padronizada, e a entidade que expressa a funcionalidade destes agentes é definida pela disponibilidade de seus serviços.

Os serviços oferecidos pelo DUA aos seus usuários com o objetivo de facilitar o acesso aos serviços de Diretório são chamados de funções locais e não são objetos de padronização. Tais serviços não requerem interação com outra entidades do sistema.

A implementação do DUA pode ser realizada na forma de programa em um sistema mono-usuário, como PC, ou um processo em um sistema multi-usuário, como mainframes ou estações de trabalho.

A estrutura do DSA pode comportar arquiteturas bem variadas. Uma delas seria a de um "mainframes" onde existe suporte para serviços internos de diretório no próprio sistema hospedeiro. Da mesma forma se enquadra um nodo comporta de uma rede local com a função de servidor dos serviços do DSA.

As possíveis evoluções da estrutura do DS sugere uma concepção de redes onde seus elementos devem apresentar flexibilidade para comportar a distribuição da funcionalidade da pópria rede.

Desta forma, o modelo funcional dos DS cobre estruturas de implementações de Sistemas de Diretório que podem ser baseados em interligações de "mainframes", em interconexões de redes e até em comunicação entre redes públicas internacionais.

3.2 Os protocolos Envolvidos

As interações entre um DUA e um DSA localizados em sistemas abertos diferentes são suportadas pelo Protocolo de Acesso de Diretório (DAP). O DAP é utilizado para efetuar os serviços de diretório. De forma similar, as interações entre DSA's pertencentes a sistemas abertos diferentes são suportadas pelo Protocolo de Sistema de Diretório (DSP). O DSP é utilizado para prover a funcionalidade de distribuição de operação. Tanto o DAP como o DSP são processos residentes na camada de aplicação OSI.

O DAP e o DSP são protocolos para prover comunicação entre um par de processos de aplicação. No ambiente OSI isto é representado como uma comunicação entre um par de Entidades de Aplicação (AE's) utilizando o serviço de apresentação. A função de uma AE é fornecida por um conjunto de Elementos de Serviços de Aplicação (ASE's). A interação entre AE's é descrita em termos da utilização por elas dos serviços fornecidos pelos ASE's.

3.3 As facilidades de Serviços do DS

Todos os serviços são providos pelo Diretório em resposta aos pedidos dos DUAs. Existem pedidos de serviços que permitem consultar o Diretório (e. g. ler, pesquisar), que permitem modificar o Diretório (e.g. remover entradas, acrescentar entradas), e aqueles serviços que podem ser qualificados (e. g. controles como prioridade, tempo).

O Diretório sempre informa o resultado da operação do solicitado. O Diretório assegura que mudanças na DIB, em consequência do resultado do pedido de Serviço de Diretório ou por algum outro meio (local), resulta em uma DIB que continua a obedecer as regras do Esquema de Diretório.

4 Desenvolvimento do Sistema de Diretório do NCE

4.1 Modelagem Funcional

O procedimento básico do funcionamento do DS consiste em um ciclo de interrogações. Este ciclo analisa a comunicação com o módulo que implementa o CASE, analisa a comunicação com o usuário, e ativa quando necessário os procedimentos de transferência das operações para o módulo CASE ou para o usuário.

O sistema pode ser dividido em dois módulos, que são:

- O módulo DUA, que contém as funções ativadas pelo usuário e que correspondem aos serviços de diretório.

  A principal função de um Agente de Usuário de Diretório é acessar o Diretório a fim de obter serviços de diretório de interesse de um usuário. Além disso, o agente de usuário deve fornecer facilidades locais ao seu usuário de forma a melhor oferecer os serviços de diretório.

  As operações de um DUA são descritas em termos das suas interações com o usuário e com o ACSE.

- O módulo DSA, cuja principal função é prover aos seus DUA's e/ou outros DSA's o acesso a DIB. Um DSA pode usar informações armazenadas em sua base de dados local ou interagir com outros DSA's para executar os pedidos.

  As operações de um DSA são descritas em termos das suas interações com o DUA e com o CASE.

4.2 Implementação

4.2.1 O Ambiente de Implementação

O ambiente de implementação adotado para os DUA's será a estação de trabalho SUN com sistema operacional tipo UNIX. Já o ambiente de implementação adotado para os DSA's constitui-se do sistema computacional VAX 8810 com sistema operacional VMS. Ambos os sistemas são multi-usuários e incorporam mecanismos para comunicação e sincronização entre processos.

4.2.2 A estrutura de Implementação

- **A Estrutura de Implementação do DUA na SUN/UNIX**

  A estrutura de implementação do DUA na SUN/UNIX será composta por vários processos que se comunicam através de troca de mensagens.

  A comunicação entre os processos na SUN será obtida através da troca de mensagens que são divididas em duas partes: a primitiva propriamente dita e os

dados de usuário. A primeira é enviada através de filas de mensagens mantidas em memória compartilhada. Nela estão contidos o deslocamento e o tamanho dos dados de usuário que também são mantidos na memória compartilhada. As filas de mensagens simulam em funcionalidade as caixas postais disponíveis no VMS da DEC onde muitos módulos do projeto foram originalmente desenvolvidos. A divisão da mensagem em duas partes também decorre daí pois o VMS limitava o tamanho máximo de uma mensagem trocada via mailbox acrescido do fato de a comunicação via caixas postais para mensagem de dados ser um processo lento na DEC.

- **A Estrutura de Implementação do DSA no VAX/VMS**

  A estrutura de implementação do DSA no VAX/VMS é composta por dois processos que se comunicam através de troca de mensagens.

  Quanto a comunicação entre os processos no VAX/VMS, existem dois tipos de mensagens: a mensagem de dados e a mensagem de sinalização.

  As mensagens de sinalização são trocadas através de caixas postais ("mailboxes"), isto é, filas FIFO do próprio Sistema Operacional. Um processo pode ter caixas postais de entrada (fila de mensagens recebidas) e caixas postais de saída (filas de mensagens enviadas). A cada caixa postal de saída está associada a identificação do processo receptor das mensagens transmitidas daquela caixa postal de saída.

  Por questões de eficência, preferiu-se realizar a transferência de dados através de áreas globais, isto é, áreas de memória comuns a todos os processos. A transferência de dados entre os processos é realizada através das caixas postais e das áreas globais, sendo a caixa postal utilizada apenas para sincronização, ou seja, somente o apontador para a área alocada para a mensagem é realmente passado pela caixa postal.

  Isto se deve ao fato de que utilizar o procedimento de comunicação de processos via caixas postais para mensagens de dados é um processo lento no ambiente em questão.

  A comunicação entre processos pode ser dividida em dois casos:

  - Comunicação de processos pertencentes a um único domínio de gerenciamento;
  - Comunicação de processos pertencentes a domínio de gerenciamento distintos.

  O primeiro caso é mais simples e se resume basicamente na troca de apontadores. A mensagem sendo trocada está armazenada na área global e apenas seu deslocamento dentro dessa área é colocada numa caixa postal de entrada do processo destino, ou seja, uma simples troca de apontadores.

  O segundo caso é mais complexo pois envolve a transmissão/recepção de unidades de dados entre os domínios de gerenciamento distintos. Para tal, é implementado o DSP descrito na recomendação X.518 do CCITT e utilização de recursos de rede.

4.3 Validação da Implementação

O processo de validação proposto para a implementação é composto de duas fases distintas. A primeira fase compreende a elaboração de ferramentas para suporte ao processo de validação. A segunda fase engloba uma avaliação experimental do comportamento dos procedimentos realizada de forma exaustiva. Nesta segunda fase é observada a relação entre os procedimentos propostos para a implementação e os procedimentos implementados efetivamente.

4.3.1 Ferramentas de Suporte à Validação

O primeiro elemento de suporte para o processo de implementação consiste na implantação de uma nova tarefa no sistema cuja principal atividade é armazenar todas as operações trocadas entre duas ou mais entidades funcionais do DS permitindo assim a obtenção de um "log" que contém todas as unidades de dados transmitidas ou recebidas para/de um outro domínio de gerenciamento. Esse log é dito ser horizontal.

O segundo elemento de suporte também consiste na implementação de uma nova tarefa no sistema cujo principal atividade é armazenar todos os eventos ocorridos no âmbito de uma submissão/recepção de operação permitindo a obtenção de um log que contém todas as informações do percurso da operação dentro de um único domínio de gerenciamento. Esse log é dito ser vertical.

O terceiro elemento consiste de um simulador da camada de sessão, que permite a instalação e interligação de duas instâncias da implementação do DS num único ambiente de implementação.

4.3.2 Processo de Validação Experimental

De posse das ferramentas descritas na seção anterior, é iniciado o processo de validação experimental da implementação, de uma forma gradual, permitindo que a complexidade embutida nesta fase seja diluída em etapas sequenciadas. É bom lembrar que antes de chegar neste estágio de validação, cada um dos módulos que contém os componentes do DS foram testados separadamente, usando os elementos de suporte apresentados acima. Este processo de validação pode ser dividido em quatro etapas:

1a. ETAPA

Nesta etapa inicial é analisado o comportamento de um domínio de gerenciamento simples formado por um único DSA e um número qualquer de DUA's para que as operações básicas de submissão e entrega possam ser verificadas sem levar em consideração os aspectos de transferência de operações entre DSA's. Pode ser testado a interface DUA/USER e o protocolo DAP.

2a. ETAPA

A segunda etapa inclui a análise de dois domínios de gerenciamento simples, ou seja, um sistema obtido pela duplicação do sistema utilizado na primeira etapa descrita acima.

A análise é feita através de mecanismos de simulação. Isto permite que o monitoramento se concentre em um único ambiente de implementação. Desta forma, são criadas duas instâncias de implementação do DS, interligados por um simulador da camada de sessão. Por meio dos elementos supervisores das suas instâncias, pode ser analisado o comportamento do sistema.

Neste ambiente é possível a simulação de falhas ocasionais, permitindo a observação do comportamento da implementação nestas condições. Pode ser testado o protocolo DSP.

3a. ETAPA

A terceira etapa consiste em transpor a configuração estabelecida na etapa anterior para dois ambientes distintos, os quais são interligados por serviços de rede. A observação do comportamento do sistema localizado remotamente também é realizada através dos elementos supervisores utilizados na segunda fase.

4a. ETAPA

Nesta última etapa, o número de domínios de gerenciamento é expandido para vários ambientes distintos, permitindo a avaliação do comportamento da implementação quanto aos aspectos relacionados ao tratamento simultâneo de diversas operações, as quais são transferidas através de unidades de dados de protocolo.

4.4 Interface Homem-Máquina

Chamamos de interface a parte do Agente de Usuário que interage diretamente com o usuário, apresentando o conjunto de serviços de diretório, bem como facilidades locais.

Em termos simples, é o programa que o usuário executa para ter acesso ao sistema de diretório.

4.4.1 A Interface e o Usuário

O objetivo da interface é prover ao usuário leigo uma ferramenta para recuperação de informações de propósito geral, sem que sejam necessários conhecimentos detalhados sobre conceitos de diretório.

A interface deverá ser amigável e ter funcionamento razoavelmente intuitivo. Para isso, ela deverá ser desenvolvida utilizando-se janelas e menus de barra, além de ter suporte para uso de mouse. Assim, o usuário poderá facilmente caminhar pelas opções oferecidas e preencher os campos necessários para a execução das operações requeridas.

O funcionamento de toda a interface deverá ser simples e totalmente ditado pelo usuário. Quando uma operação for selecionada posicionando-se a barra na opção correspondente e clicando-se o primeiro botão do mouse, a função que a executará será acionada; novas janelas serão apresentadas sobrepostas à janela principal para que o usuário preencha os parâmetros referentes a operação.

O usuário não precisará preencher todos os parâmetros de uma operação, desde que estes não sejam obrigatórios. Além disso, muitos dos parâmetros já possuem valores "defaults", ficando a critério do usuário modificá-los. Os parâmetros cujo domínio de valores é limitado, por exemplo a SIM ou NÃO, serão alterados apenas clicando-se o botão esquerdo do mouse sobre a opção correspondente.

Ao contrário de outras interfaces de sistemas para recuperação de informações, esta oferecerá acesso a uma grande variedade de tipos diferentes de informação. Por isso, a interface DUA de uso geral deverá suprir o usuário com o necessário conhecimento sobre os tipos de informações que ele pode encontrar no Diretório. Esta facilidade será oferecida através da opção de "navegação" pela Árvore de Informações de Diretório (DIT).

Na implementação de janelas, menus de barras e, para controlar todo fluxo de informações entre o usuário e o sistema, será utilizado o pacote de rotinas OpenWindows.

Este pacote oferece poderosos recursos para a construção de interfaces baseadas em X-Windows.

Para administrar os arquivos de dados do Diretório e, de um modo geral, todo sistema de armazenamento de informações em memória secundária, tais como log e nome amigável, serão usadas as rotinas da biblioteca stdio da linguagem C.

5 Considerações Finais

A evolução do nosso DS deve se basear nos resultados experimentais obtidos durante os processos de validação e avaliação da proposta de implementação, visando a perfeita adequação entre os serviços e os anseios por parte da comunidade de usuários.

No instante em que a implementação atingir um grau de maturidade correspondente ao desejado, deve-se estimular e facilitar a sua transposição para outros ambientes de implementação.

6 Referências

[1] Benford S., "Navigation and Knowledge Management within a Distribuited Directory System", Proceeding of the IFIP WG 6.5 Working Conference on Message Handling System and Distributed Applications, Costa Mesa, California, U.S.A., 1988, pp 115-134

[2] Huitema C., "Using a standard Directory as a Name Server within the Thorn Project", Proceeding of the IFIP WG 6.5 Working Conference on Message Handling System and Distributed Applications, Costa Mesa, California, U.S.A., 1988, pp 135-144

[3] Kille S., "The QUIPU Directory Service", Proceeding of the IFIP WG 6.5 Working Conference on Message Handling System and Distributed Applications, Costa Mesa, California, U.S.A., 1988, pp 145-158

[4] CCITT Recommendations X500 Series: The Directory System, 1988, Blue Book, vol VIII Fascicle VIII.8.

[5] CCITT X217 Recommendation:Association Control Service Definition for CCITT Applications.

[6] CCITT X219 Recommendation:Remote Operations:Model, Notation and Service Definition.

[7] Pirmez l., Mendes S., O Diretório X500: Uma solução para o problema de resolução de endereços no MHS", IX Simpósio Brasileiro de Redes de Computadores,Florianópolis, SC, Maio de 1991.

Modelagem e Animação de Superfícies Deformáveis

Luciana Porcher Nedel*

Instituto de Informática

Universidade Federal do Rio Grande do Sul

Caixa Postal 15064 - CEP 91501

Porto Alegre - RS - Brasil

e-mail: lpn@inf.ufrgs.br

RESUMO

O crescente número de sistemas de animação baseados na cinemática para geração do movimento vem levando os pesquisadores a buscar outras alternativas para gerar resultados mais realistas. Com base nesta premissa, vários autores passaram a estudar a criação de animação de objetos sintéticos através da dinâmica. Este trabalho apresenta um modelo simplificado para simulação do movimento de uma superfície flexível. São abordados tanto o modelo geométrico utilizado como o modelo físico, sendo ressaltadas as forças aplicadas sobre a superfície e as restrições que podem ser impostas pelo mundo no qual a mesma está inserida.

PALAVRAS-CHAVE: Animação, Deformação, Dinâmica, Elasticidade, Restrições, Simulação

1 INTRODUÇÃO

Nos últimos anos, tem havido um interesse crescente em desenvolver modelos baseados na Física para gerar seqüências de animação [5]. Num primeiro momento foram abordados apenas objetos rígidos, passando-se mais tarde a considerar objetos articulados e, por fim, aqueles ditos flexíveis ou, ainda, deformáveis.

A animação de personagens deformáveis exige que se possua um modelo geométrico que permita a mudança de forma ao longo do tempo. Em Computação Gráfica, isto pode ser

* Atualmente integra o corpo docente da Universidade Luterana do Brasil (ULBRA)

definido através de modelos físicos de curvas, superfícies ou mesmo sólidos deformáveis, governados através de leis mecânicas. Essas leis podem ser expressas, por exemplo, através de equações diferenciais dinâmicas, unificando, deste modo, a descrição de formas e movimentos. Com a resolução dessas equações numericamente, devemos ser capazes de criar animações realísticas envolvendo a interação entre modelos deformáveis e a aplicação de várias forças, meio ambiente e obstáculos impenetráveis, em uma simulação física da realidade [4].

O objetivo principal na simulação de modelos deformáveis ou flexíveis é produzir movimentos fisicamente realistas. Exemplos incluem a simulação da musculatura do corpo humano a fim de representar realisticamente um personagem caminhando; a simulação do fluxo de líquidos viscosos, como, por exemplo, lavas sobre rochas vulcânicas; a simulação de um escultor modelando com argila [2]; ou ainda a simulação de tecidos num sistema de CAD para a indústria de confecções [1][6].

Neste trabalho, apresenta-se um método de modelagem e animação de superfícies deformáveis, que permite tanto a simulação do comportamento estático e dinâmico de uma superfície, como a sua interação com um mundo virtual.

2 MODELO GEOMÉTRICO

O modelo geométrico aqui considerado para simular uma superfície deformável consiste de uma malha retangular representada por uma matriz de pontos M. Cada elemento Mi,j desta matriz contém um terno (x,y,z) correspondente a um ponto da superfície, em um espaço tridimensional. Considera-se ainda r como a distância entre os elementos da matriz M acima definida.

3 MODELO FÍSICO

A fim de simular fisicamente uma malha flexível, utilizou-se o conceito das molas de Hooke [3]. O modelo físico baseia-se fundamentalmente na aplicação de forças sobre os diversos pontos de massa da malha, gerando novas posições para os mesmos. Através da

soma de todas as forças aplicadas, obtém-se uma força resultante para cada ponto da superfície deformável.

Com o objetivo de simplificação, considerou-se aqui a aplicação de apenas três forças distintas: força da gravidade, força de elasticidade e força de curvatura e torção; além de uma quarta força, representando as restrições do sistema. Desta forma, pode-se dizer que a força resultante em cada ponto da malha pode ser calculada da seguinte forma:

$$F_{RESULTANTE} = F_{GRAVIDADE} + F_{ELASTICIDADE} + F_{CURVATURA} + F_{RESTRIÇÃO}$$

Isto, no entanto, não quer dizer que não possam ser aplicadas outras forças ao modelo. Por exemplo, para incluir a simulação da força do vento, basta acrescentar este elemento na soma referida acima.

3.1 FORÇA GRAVITACIONAL

A força da gravidade é calculada para cada ponto da superfície, da seguinte forma:

$$F_{GRAVIDADE} = g.m$$

onde g representa a aceleração da gravidade e m, a densidade de massa do ponto.

3.2 FORÇA DE ELASTICIDADE

Para simular a elasticidade em um ponto Pi, j da matriz Mi, j, supôs-se a conexão deste aos seus 8 vizinhos, através de molas (figura 1). Sabendo-se que a força exercida por uma mola de Hooke sobre um ponto de massa é

$$F_{MOLA} = -km \cdot (P - Pr)$$

onde km é a constante que indica o grau de elasticidade da mola, P, o ponto em questão e Pr o ponto de repouso da mola, pode-se, por extensão, definir a força de elasticidade como sendo o somatório das forças exercidas pelas molas, no ponto P.

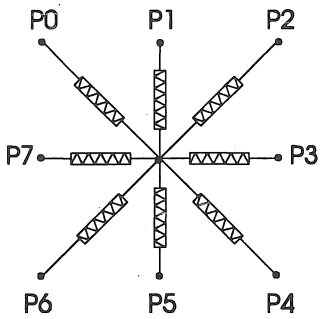


Figura 1: Modelo elástico da superfície

3.3 FORÇA DE CURVATURA E TORÇÃO

A força aqui descrita indica o grau com que a superfície se curva e torce. Esta força é calculada também em função dos 8 vizinhos de um ponto. A simulação deste efeito foi concebida como a colocação de molas angulares entre os vizinhos do ponto P, da seguinte forma: uma mola no ângulo formado entre os vértices $P0, P$ e $P4$ (figura 2); uma entre $P1, P$ e $P5$; outra entre $P2, P$ e $P6$ e uma última entre $P3, P$ e $P7$.

A colocação destas molas é feita de forma que estejam na posição de repouso quando o ângulo formado entre os 3 vértices for igual a 180^o. Além disto, considera-se que as molas em questão possuam um coeficiente de elasticidade parametrizável, denominado kc. A alteração que a força exercida por estas molas possa vir a causar na posição do ponto Pi, j, é indicada por um função $f(i, j)$.

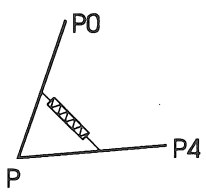


Figura 2: Mola angular entre os pontos $P0, P$ e $P4$, com ângulo inferior a 180^o.

4 RESTRIÇÕES

No que diz respeito às restrições físicas impostas pelo mundo virtual no qual a superfície está sendo simulada, considerou-se:

- a existência de pontos imóveis na grade que representa a superfície;
- a detecção de colisão de superfícies deformáveis com objetos sólidos.

Entretanto, a concepção do modelo aqui descrito, por ser baseada em restrições físicas, permite a inclusão de novas restrições, sem que, no entanto, isto acarrete modificações drásticas no modelo existente.

A implementação destas funções é feita da seguinte forma: no caso de haver qualquer tipo de restrição sobre um ponto, é calculada uma força tal que leve o ponto a não se mover ou, ainda, a se mover obedecendo a restrição em questão.

4.1 PONTOS FIXOS

Trata-se de uma restrição imposta ao sistema ainda na fase de configuração, ou seja, no instante zero. Através da indicação de quais os pontos da malha que devem permanecer fixos, pode-se simular situações em que a superfície mantém-se "pendurada" por um ou mais pontos. Esta restrição pode ser usada, por exemplo, na simulação de uma toalha pendurada em um prego; uma cortina presa na parede; ou ainda, uma bandeira presa a um mastro e tremulando ao vento.

4.2 COLISÃO COM CORPOS RÍGIDOS

A colisão da superfície deformável com corpos rígidos, é calculada como uma função $f(i,j)$ que verifica para cada ponto da malha se este está dentro do sólido, fora ou na superfície deste. No caso de estar fora ou sobre a superfície, a força de restrição é nula. Entretanto, se o ponto da superfície estiver dentro do sólido, deve ser aplicada uma força que leve o ponto para a superfície do mesmo.

Com o uso desta restrição, pode-se gerar efeitos bastante interessantes como a colocação de uma toalha sobre uma mesa.

5 ANIMAÇÃO

A simulação de movimento no modelo proposto, dá-se através da aplicação dos conceitos conhecidos da dinâmica [3]. Abaixo, descreve-se o algoritmo utilizado na geração dos pontos da malha, após cada intervalo de tempo especificado (Δt):

- Para um tempo igual a t , inicialmente igual a 0 e variando de Δt:
  - Calcular a força resultante em cada ponto da malha flexível, através dos métodos acima descritos;
  - Conhecidas as forças resultantes e a densidade de massa de cada ponto, através da aplicação das leis de Newton ($F = m.a$, onde m é a massa e a, a aceleração) calcular a aceleração de cada ponto.
  - Com base nesta aceleração, encontrar a velocidade dos pontos através da fórmula: $V = Va + a.\Delta t$, onte Va representa a velocidade no momento anterior, a, a aceleração e Δt, o intervalo de tempo.
  - Dada a velocidade, encontra-se a nova posição do ponto, da seguinte forma: $P = Pa + V.\Delta t$, onde Pa é a posição anterior do ponto, V , a sua velocidade e Δt, o intervalo de tempo.

6 VALIDAÇÃO DO MODELO PROPOSTO: RESULTADOS INICIAIS

Foi desenvolvido um sistema considerando uma malha deformável e apenas duas dimensões. Na figura 3, apresenta-se 3 momentos da deformação de uma superfície bastante elástica (constante km igual a 1), leve (densidade de massa igual a 1), sem resistência a curvatura e torção e presa apenas por um ponto.

O modelo proposto está inserido no âmbito da dissertação de mestrado da autora e foi desenvolvido com o objetivo de constituir um passo inicial às pesquisas na área da deformação de objetos através da Física. Com base nesta premissa, pode-se considerar que os objetivos foram alcançados, uma vez que o modelo possui um grau de realismo bastante razoável.

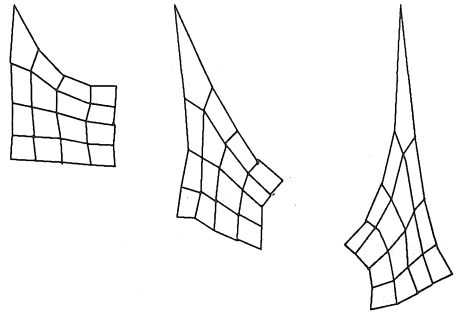


Figura 3: Deformação de uma superfície de 5 x 5.

7 CONCLUSÃO: PERSPECTIVAS FUTURAS

Conforme foi afirmado anteriormente, o presente modelo possui um grau de realismo tal que permite o prosseguimento do trabalho de acordo com várias propostas. A primeira delas é a extensão destes conceitos para gerar não apenas superfícies deformáveis, mas também objetos tri-dimensionais compostos por faces, com essas mesmas características.

Uma segunda possibilidade de extensão deste trabalho é a inclusão de características inelásticas ao modelo. Entre elas, pode-se considerar a deformação definitiva das molas que unem os pontos da malha ou, ainda, a fratura. Entende-se por fratura o rompimento das molas por diferentes motivos, sendo o mais evidente a distensão demasiada de uma superfície pouco resistente. Formalmente representa-se a resistência de um material, através da constante km das molas. Quanto maior esta constante, maior a resistência das molas; conseqüentemente, também menor a elasticidade da superfície.

Outra linha de pesquisa a ser desenvolvida é a criação de superfícies que em seu estado inicial tenham forma diferente de um retângulo, como o proposto neste artigo. Para isso, imaginou-se a criação de um modelador que gerasse superfícies de formatos diferentes e que respeitasse as seguintes restrições: a superfície deve poder ser representada em uma matriz Mi, j; as molas que compõem o sistema devem estar todas em repouso, no estado inicial.

Em relação a este último aspecto - restrições, cabe salientar que o presente trabalho se insere no contexto de desenvolvimento de sistemas baseados em restrições e, sob este ponto

de vista, uma alternativa viável no prosseguimento da pesquisa é a estruturação do sistema de modelagem e animação segundo a abordagem orientada a objetos.

AGRADECIMENTOS

Ao professor Anatólio Laschuk, pelo auxílio na definição de um modelo de molas adequado; à professora Carla Freitas, pela revisão final do texto; ao Niuton Zinn Severo e a Soraia Raupp Musse pelo apoio e incentivo; ao CNPq por financiar parcialmente este trabalho.

BIBLIOGRAFIA

[1] Dias, J.M.S. e Rebordão, J.M., "Um Modelo Energético para Simulação de Superfícies Deformáveis", Actas do IV Encontro Português de Computação Gráfica, pp.109-123, Lisboa, Portugal, Nov. 91.

[2] Platt, J.C. e Barr, A.H., "Constraint Methods for Flexible Models: the Tutorial Notes", Developments in Physically-Based Modeling - SIGGRAPH 88 - COURSE 27, L1-L23, 1988.

[3] Resnick, R. e Halliday, D., Física I, Ed. Livros Técnicos e Científicos Ltda., 1984.

[4] Terzopoulos, D. e Fleischer, K., "Deformable Models", The Visual Computer, 4:306-331, Springer-Verlag, 1988.

[5] Terzopoulos, D. e Witkin, A., "Physically-Based Models with Rigid and Deformable Components", Graphics Interface, pp.146-154, Edmondon, Alberta, Jun. 88.

[6] Weil, J., "The Synthesis of Cloth Objects", Computer Graphics, 20(4):49-54, Aug. 86.

MHC - A Connectionist architecture for hybrid training

José M. Ramírez
Paradigma C.A.
Apartado 67079, Caracas 1061
Venezuela
Phone: +58-2-2836942, Fax: +58-2-2832689

Abstract

We present a Connectionist architecture called Modified Hyperspherical Classifier (MHC) based on: 1) the work of Cooper [5][6] and Batchelor [2][3][4] about Hyperspherical Classifiers, 2) the RCE paradigm as described by Scofield et al. [10] and 3) some new considerations derived from the search of an efficient model to perform heterogeneous pattern processing using supervised and/or unsupervised training algorithms depending on the nature of the problem, the availability of the correct output during the training process and the operational state of the network. We use the term Hybrid Training to define the use of a supervised or an unsupervised strategy to train the same network; this definition differs from the presented by Hertz et al. [7] as Hybrid Learning, which refers to layered networks with different learning strategies for each layer (e.g. Counterpropagation).

Conclusions about the performance of the architecture are presented, based on the execution of tests using tasks related with Familiarity detection, Principal component analysis, Clustering, Prototyping, Feature mapping and Pattern transformation.

Keywords: Connectionist Architectures, Classifiers, Neural Networks.

HC, RCE and MDC

Like the nearest-neighbor classifier, the HC is based upon the storage of patterns that represent points in a space. The association of an unknown pattern with a known category is made by the distance function (Cartesian, Hamming, etc.). The main difference with the nearest-neighbor model is that each point has a region of influence that is defined by a sphere with the pattern´s location point as the center. Each stored pattern with its region of influence defines a decision region associated with the category or class of the stored point.

RCE networks can be viewed as special cases of HC in that the stored patterns are stationary in the space and the regions of influence can only shrink and can not expand, this approach is also known as DSND (Disjoint Spheres/ No Drift) because the training process try to disjoint the spheres of different categories to avoid confusion in the classification but the stored point, the center of each sphere, remains stationary.

The MDC model uses the N-dimensional feature space as RCE and HC, but its functionality is based on the definition of at least one prototypical point for each category to be considered, this definition is made in the initialization stage. The classification is performed based on a distance metric between the prototypes and the pattern being processed. details of MDC can be found in [10],[12] and [13].

Modified Hyperspherical Classifier

The proposed architecture (MHC) has the following properties:

1. High storage density based on the N-dimensional feature space model.

2. The stored points (hidden layer units) act like discriminat functions.

3. The connections between the input and hidden layer are weighted. The weights represent the relative importance of each feature (input unit) to classification . This importance can be set "a priori" or by an analysis (e.g. covariance) of the input patterns.

4. The hidden units contain the location of the stored points associated and the size of the region of influence (radius of the sphere).

5. The connections between the hidden layer and the output layer are weighted. the weights are the result of a probabilistic density function applied to the region of influence of the unit. If an unit H1 shares part of the space with an unit H2 of another category, the weight of the connections of H1 and H2 with the corresponding output units will be <1 and will depend on the probability that a point in the shared region belongs to H1´s or H2´s category. The density function can be easily implemented as a count of the correct classifications performed over patterns that "fires" both neurons H1 and H2.

6. The spheres can shrink and expand during the training process to form the category areas in the feature space. Several spheres can be summarized into one or an sphere can be moved to a different location.

7. All the adjustments concerning the classification of the pattern being processed are made before processing the next pattern. The convergence of the net is always reached in just 2 epochs.

8. The initial radius of the spheres are not initially set to a fixed default value, but to the maximum value that can be assigned without include a point of a different category.

Figure 1. illustrates a MHC network

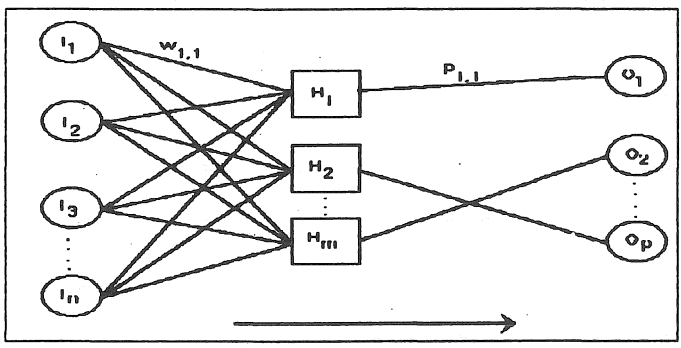


Figure 1.

Each input unit corresponds to a feature and each output unit corresponds to a pattern category. The hidden units are the stored points (spheres).

The key characteristics of MHC are: Feedforward, High storage density, Reduced connectivity, Partially distributed, Dynamic category learning and hybrid training.

Operation and Training of the MHC

When a pattern is presented to the network the activation of the hidden units will be given by:

$$H_j = G((SW_{ij}*(X_i-P_i)^2)^{1/2})- T)$$

$$G(x) = \quad 0 \text{ if } x > 0$$

1 if x <= 0

where:

H_j is the activation of the jth hidden layer

X_i is the ith feature of the input pattern

P_i is the ith component of the stored point

W_{ij} is the connection between the ith input unit and the jth hidden unit

T is the threshold or radius of the sphere (the region of influence)

The activation of the output units will be given by:

$$O_j = H_i * P_{ij}$$

where:

P_{ij} is the weight of the connection between the hidden unit and the associated output unit. Corresponding to the probabilistic density function.

The function G(x) associates each hidden unit with a region in the feature space. The location of the region is stored in the hidden unit and the size of the region is determined by the threshold also stored in the hidden unit. Any input pattern falling within the region of influence of a given unit will cause the unit to transfer output to the corresponding output unit. A pattern will cause the firing of all the units that share the region where the pattern

is located, if the hidden units project theirs output to more than one output unit (categories) the result will be an "ambiguous" classification; if only one output unit fires the answer of the network will be "known", the other possible output is "unknown", this results when no output unit fires.

The training process of MHC networks involves the execution of several actions: a) Creation of hidden units (spheres) and/or output units (categories). b) Adjustment of the spheres, traduced in shrinking, expansion, movement or jointing and c) Adjustment of the hidden to output connections using a probabilistic density function.

The conditions to perform the above actions are different in each training strategy. The control over their execution resides in a preprocessing module that determines if the pattern is prepared (or intended) for unsupervised or supervised training.

Hybrid training is necessary since there are problems where the solution is poorly known or just a subset of the possible categories are needed; in this case the network can be trained in a supervised way using the patterns which solution is known or just with patterns representing the desired categories. This process initializes the network and then a unsupervised training can take place until the entire feature space is covered. If the solution of a problem is totally unknown (e.g. exploratory data analysis) the network can be trained in an unsupervised way, using all the patterns available. The network will form clusters that can be analyzed for the experts in the domain and the categories can be defined to proceed with supervised training over the same network or a new one.

In the supervised strategy, if the output of the network is "unknown", a new hidden unit is created using the location of the current pattern and this unit is associated to the output unit of the correct category, if the category is not in the network (there is no output unit for this category) a new output unit is created, this process is known as dynamic category learning. the weight of the connection between the hidden unit and the output unit will be 1 and the size of the region of influence (threshold) will be the maximum value that does not include a point of another category. In an "ambiguous" situation, the spheres that are not associated to the correct category are shrunk so that they no longer cover the current pattern. If that size of the spheres reached the minimum value and remains the overlapping with spheres of another category, the probabilistic density function is used to adjust the connections between the overlapping units and the associated output units.

In the unsupervised strategy, if the output of the network is "unknown" a new category is created (hidden unit and output unit) using the same procedure described above, the category identifier is generated by the network. If more that one output unit fires, the categories involved are ranked according to the computed degree of belonging (as in MDC). The initial size of the spheres in the unsupervised strategy is settled to a default maximum value given by the user, obviously the maximum possible size can not be used as in the supervised strategy.

Results and Conclusions

1. The MHC adapt to a wide range of applications, since there´s no restriction derived from the size of the problem (storage requirement) or nature (unsupervised or supervised).

2. The convergence time is reduced to 2 epochs, no matter what training strategy is used.

3. The design of the network is simplified, due to the evolving nature of the model. The network designs itself.

4. The HMC is able to learn to separate very complex decision surface.

5. The hardware implementation of HMC is easy, compared to parametric feedforward networks (e.g., backpropagation).

Acknowledgments

Thanks to my colleague Mauricio Paletta who wrote most of the C++ code for RCE and also the base code for MDC.

References

[1] J. A. Anderson et al. "Categorization and Selective Neurons". Parallel Models of Associative Memory. pp. 213-236. 1.981. Earl Baum. N.J.

[2] B. Batchelor, B. Wilkins. "Adaptive Discriminant Functions". Pattern Recognition. 1.968 IEE Conf. Publication, 42, pp 168-178.

[3] B. Batchelor. "Learning machines for pattern recognition". Ph.D. Dissertation, Southampton.

[4] B. Batchelor. "Practical approach to pattern classification". 1.974. New York. Plenum Press.

[5] P. Cooper. "The Hypersphere in pattern recognition". 1.962. Information and Control, 5, pp. 324-346.

[6] P. Cooper. "A note on an adaptive hypersphere decision boundary". 1.966. IEEE Transactions on Electronic Computers, pp. 948-959.

[7] J. Hertz et al. "Introduction to the theory of Neural Computation", Addison-Wesley, 1.991.

[8] S. Grossberg. "Adaptative Pattern Classification and Universal Recoding: Part I". Parallel Development and Coding of Neural Feature Detectors. Biological Cybernetics. Vol. 23. pp. 121-134. 1.976.

[9] T. Kohonen. "Clustering, Taxonomy and Topological Maps of Patterns". Proceedings of the sixth International Conference of Pattern Recognition. pg. 114-125. 1.982. IEEE Press.

[10] **N. Nilsson.** "Mathematical Foundations of Learning Machines". 1.990. Morgan-Kaufmann.

[11] **J. Ramírez.** "Connectionist Learning Methods". Tutorial in the 1st. World Congress on Expert Systems Dec. 1.991. Orlando, Florida.

[12] **J. Ramírez.** "Use of Connectionist Classifiers to detect regularities in data". 1.991. Intevep S.A. tech. report.

[13] **J. Ramírez, I. Torres.** "Conclusions of the use of connectionist models as feature detectors", Intevep S.A., 1.991. tech. report.

[14] **J. Ramírez, I. Torres.** "Connectionist architectures for real-time pattern processing: Design issues", Intevep S.A., 1.991. tech. report.

[15] **D. Reilly et al.** "A neural model for category learning". 1.982. Biological Cybernetics, 45, pp. 35-41.

[16] **D. Reilly et al.** "Learning system architectures composed of multiple learning modules". 1.987. Proc. IEEE First International Conference on Neural Networks, San Diego, CA, June 21-24.

[17] **D. Reilly, L. Cooper.** "An overview of neural networks: early models to real world systems". 1.990. An introduction to Neural and Electronic Networks. Academic Press.

[18] **D. Rumelhart, J. Mc Clelland.** "Parallel Distributed Processing Explorations in the Microstructure of Cognition". Vol. I, II & III. 1.988. MIT Press.

[19] T. Sanger, "Optimal Unsupervised Learning in a Single-layer Linear Feedforward Neural Network", Neural Networks, vol. 2 pp. 459-473, 1.989.

[20] C. Scofield et al. "Pattern class degeneracy in an unrestricted storage density memory". 1.988. Neural Information Processing Systems, pp. 674-682. New York American Institute of Physics.

[21] C. Scofield. "Learning Internal Representations in the Coulomb Energy Network", Cybernetics, 1.987.

Derivaciones Perezosas en Programación Lógica con Restricciones Ecuacionales*

María José Ramis ‡ Javier Piris ‡ María Alpuente ‡

Resumen

El lenguaje CLP($\mathcal{H}/\mathcal{E}$) es una instancia del esquema CLP [14] que resuelve ecuaciones bajo una teoría de la igualdad presentada como una teoría ecuacional de Horn $\mathcal{E}$ [1,2]. CLP($\mathcal{H}/\mathcal{E}$) aúna el estilo de programación lógica basado en cláusulas de Horn, el paradigma basado en ecuaciones (condicionales) y la programación con restricciones. Como núcleo de su semántica operacional, en [1,2] se define un procedimiento incremental que semidecide la solubilidad de las ecuaciones en la estructura $\mathcal{H}/\mathcal{E}$ y que está basado en un cálculo de narrowing. La terminación de este cálculo no está garantizada, por el hecho de que el conjunto de $\mathcal{E}$-unificadores de un par de términos es sólo semidecidible. En este trabajo se considera una versión perezosa del lenguaje CLP($\mathcal{H}/\mathcal{E}$) que retrasa el problema de la $\mathcal{E}$-unificación de las ecuaciones hasta haber resuelto completamente el resto de átomos en el objetivo. Para optimizar los algoritmos se propone un conjunto de transformaciones sintácticas que simplifica las restricciones y, en ocasiones, es capaz de detectar su insatisfacibilidad.

Palabras clave: Programación lógica con restricciones, Programación lógica ecuacional, Sistemas de Reescritura, Unificación Universal, Resolución perezosa.

1 Introducción

Uno de los problemas que, en los últimos años, han atraído con más fuerza el interés de los investigadores en el campo de la Programación Lógica es el de la integración de dos de las familias más prometedoras de lenguajes declarativos: los lenguajes lógicos y los ecuacionales [5,6,8,12,15]. Recientemente, el paradigma de Programación lógica pura ha sido generalizado a un contexto más general de Programación lógica con restricciones (Constraint Logic Programming, CLP), un esquema genérico para extensiones conservativas a la programación lógica pura [11,14]. El estilo CLP se muestra especialmente fecundo en el campo de la integración de lenguajes. La integración de la programación lógica y ecuacional en el marco de CLP resulta ser simple y elegante, especialmente por la garantía de una semántica análoga a la de la programación lógica pura. El lenguaje CLP($\mathcal{X}$) resultante de una instanciación del esquema no sólo hereda las propiedades semánticas más atractivas de la programación lógica (semántica declarativa por modelo mínimo, caracterización por punto fijo y semántica operacional equivalente) sino que, además, posee la ventaja de contar con una semántica algebraica que se define directamente sobre la estructura $\mathcal{X}$ con la que se parametriza el esquema. El concepto de estructura especifica el dominio sobre el que se efectúa la computación dando la interpretación semántica a funciones y relaciones y es el elemento clave para la definición semántica algebraica del lenguaje.

El lenguaje CLP($\mathcal{H}/\mathcal{E}$) es una instancia del esquema CLP que usa ecuaciones para expresar restricciones entre símbolos de función. La estructura con la que se parametriza el esquema es el algebra cociente $\mathcal{H}/\mathcal{E}$, i.e. la partición más fina inducida por una teoría de la igualdad $\mathcal{E}$ sobre el Universo de Herbrand $\mathcal{H}$ para el programa. El único símbolo de predicado para las restricciones es el $=$, que se interpreta como igualdad semántica en el dominio.

Una cuestión fundamental a afrontar en el diseño de un lenguaje CLP es la selección de los algoritmos adecuados para la verificación de la solubilidad de las restricciones. En $\mathcal{H}/\mathcal{E}$ este problema es sólo (semi)-decidible. Como núcleo del mecanismo de resolución de restricciones que semidecide la solubilidad de las mismas, en [1,2] se define un procedimiento de *narrowing* [12,21], cuyo uso presenta como inconveniente la posibilidad de no terminación de éste cuando la restricción no es consistente. Como consecuencia, un

*Este trabajo ha sido parcialmente subvencionado por CICYT *TIC 91-0425* y realizado mientras la primera autora está disfrutando de una beca de Investigación de la Consellería de Cultura, Educación y Ciencia de la Generalitat Valenciana.

‡Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera s/n, Apdo. 22012, 46020 Valencia, Spain. *e-mail:* mjrg(jpiris)(maria)@dsic.upv.es

paso de derivación CLP($\mathcal{H}/\mathcal{E}$) puede conducir a computaciones que entran en ciclo sin producir ninguna respuesta y es para evitar esta dificultad por lo que en este trabajo se considera una versión perezosa del lenguaje. La idea de la evaluación perezosa está estrechamente relacionada con la de hacer todo lo que es seguro en cada momento, evitando el análisis de situaciones peligrosas.

La organización del trabajo es como sigue. Con la intención de ofrecer un trabajo autocontenido, en la sección segunda se revisan algunas nociones y resultados básicos sobre sistemas de reescritura condicionales y unificación semántica. También se reformulan algunos conceptos básicos del marco conceptual CLP en términos de transiciones y de configuraciones para facilitar el posterior uso de la aproximación estructural de Plotkin a la definición semántica operacional de CLP($\mathcal{H}/\mathcal{E}$) y se revisa la definición de la sintaxis del lenguaje. En la sección tercera formalizamos, utilizando la aproximación de Plotkin, nuestra propuesta de mecanismo computacional perezoso, que hace uso de un conjunto de transformaciones sintácticas para optimizar los cálculos [9,20]. Finalmente, se incluye como anexo un intérprete Prolog para el lenguaje. Asumimos en el lector un conocimiento básico de la programación lógica [19], programación lógica con restricciones (CLP) [14,11], ecuaciones y sistemas de reescritura condicionales [17] y unificación semántica [24]. En [4] se estudia la aplicación del lenguaje CLP($\mathcal{H}/\mathcal{E}$) al prototipado automático de aplicaciones de bases de datos.

2 Preliminares

Por Σ, Π y V (posiblemente con subíndices) denotamos, respectivamente, colecciones disjuntas numerables de símbolos de función, símbolos de predicado y símbolos de variable con sus signaturas. $\tau(\Sigma \cup V)$ y $\tau(\Sigma)$ denotan, respectivamente, los conjuntos de términos y términos sin variables (términos básicos) construidos sobre Σ y V. Un (Π, Σ)-átomo es una expresión de la forma $p(t_1, \ldots, t_n)$ donde $p \in \Pi$ es n-ario y $t_i \in \tau(\Sigma \cup V)$, $i = 1, \ldots, n$. Una (Π, Σ)-restricción es un conjunto (posiblemente vacío) de (Π, Σ)-átomos. El símbolo $\tilde{}$ denotará una secuencia finita de símbolos. $\tau(\Sigma)$ suele ser llamado el Universo de Herbrand ($\mathcal{H}$) sobre el alfabeto Σ.

Una Σ-*ecuación* $s = t$ es un par de términos $s, t \in \tau(\Sigma \cup V)$. Una Σ-*teoría* ecuacional de Horn $\mathcal{E}$ consiste en un conjunto finito de cláusulas de Horn ecuacionales, de la forma $e \leftarrow e_1, e_2, \ldots, e_n, n \geq 0$, donde $e, e_1, e_2, \ldots, e_n$ son Σ-ecuaciones. Una teoría ecuacional de Horn $\mathcal{E}$ puede verse como un sistema de reescritura $\mathcal{R}$ donde las reglas son las cabezas y las condiciones son los respectivos cuerpos. Representamos las ocurrencias de un término como secuencias, posiblemente vacías, de naturales; t/u es el subtérmino de t a la ocurrencia u; $t[u \leftarrow r]$ es el término t con el subtérmino a la ocurrencia u reemplazado por r; $\mathcal{V}(t)$ denota el conjunto de variables de un término t [17].

Sea $\mathcal{E}$ una teoría ecuacional de Horn. Se dice que el término s se reescribe al término t, $s \rightarrow_{\mathcal{R}} t$, si existe una (variante de una) cláusula $(l = r \Leftarrow s_1 = t_1, \ldots, s_n = t_n)$ en $\mathcal{E}$, una ocurrencia u de s y una sustitución θ tales que $s/u = l\theta$, $t = s[u \leftarrow r\theta]$ y $\forall i.\ 1 \leq i \leq n.\ \exists \gamma_i$ tal que $s_i\theta \rightarrow^*_{\mathcal{R}} \gamma_i \wedge t_i\theta \rightarrow^*_{\mathcal{R}} \gamma_i$.

Se dice que dos términos s y t son convergentes, $s \downarrow_{\mathcal{R}} t$, si existe un término ω tal que $s \rightarrow^*_{\mathcal{R}} \omega \wedge t \rightarrow^*_{\mathcal{R}} \omega$. La relación $\Rightarrow_{\mathcal{R}}$ se dice que es *confluente* si para todo término s_1, s_2, s_3 tales que $s_1 \rightarrow^*_{\mathcal{R}} s_2 \wedge s_1 \rightarrow^*_{\mathcal{R}} s_3$, existe un término s tal que $s_2 \rightarrow^*_{\mathcal{R}} s \wedge s_3 \rightarrow^*_{\mathcal{R}} s$. La relación $\rightarrow_{\mathcal{R}}$ es *noetheriana* si no existe una cadena infinita $s_1 \rightarrow_{\mathcal{R}} s_2 \rightarrow_{\mathcal{R}} s_3 \rightarrow_{\mathcal{R}} \ldots$ de reescrituras. Una teoría ecuacional de Horn se dice que es *canónica* sii la relación $\rightarrow_{\mathcal{R}}$ es confluente y noetheriana.

Cada teoría ecuacional de Horn $\mathcal{E}$ genera una relación de congruencia más fina $=_{\mathcal{E}}$ (llamada $\mathcal{E}$-*igualdad*) sobre $\tau(\Sigma \cup V)$ (la menor teoría que contiene todos los pares $s = t$ tales que[1] $\mathcal{E} \models s = t$). Se dice que $\mathcal{E}$ es una presentación o axiomatización de la teoría de la igualdad $=_{\mathcal{E}}$ y, por abuso, se habla a veces de la teoría de la igualdad $\mathcal{E}$ para denotar la teoría axiomatizada por $\mathcal{E}$. Denotaremos como $\mathcal{H}/\mathcal{E}$ la partición más fina $\tau(\Sigma)/=_{\mathcal{E}}$ inducida por $=_{\mathcal{E}}$ sobre el conjunto de términos básicos $\tau(\Sigma)$.

Se dice que dos términos s, t son $\mathcal{E}$-unificables (o $\mathcal{E}$-iguales) sii existe una sustitución σ tal que $s\sigma$ y $t\sigma$ están en la congruencia $=_{\mathcal{E}}$, i.e. tal que $\mathcal{E} \models s\sigma = t\sigma$. La sustitución σ es llamada un $\mathcal{E}$-unificador de s y t. Se llama $\mathcal{E}$-unificación al proceso de resolver una ecuación bajo la teoría ecuacional $\mathcal{E}$. Ya que el proceso es sólo semidecidible, un algoritmo de $\mathcal{E}$-unificación puede verse como un procedimiento de semidecisión para verificar la solubilidad de restricciones ecuacionales sobre el cociente $\mathcal{H}/\mathcal{E}$. Cada instancia de un $\mathcal{E}$-unificador representa una solución sobre este cociente. Las aproximaciones de mayor significación al problema de computar el conjunto de $\mathcal{E}$-unificadores de dos términos son tres: flat SLD resolución [6,10,25], conjuntos completos de transformaciones [9,12,20] y paramodulación [22] o formas especiales de ésta, tales como superposición [8] o narrowing [13,21].

Sea $\mathcal{E}$ una teoría ecuacional de Horn. Un símbolo de función $f \in \Sigma$ se dice que es *irreducible* sii no existe ninguna cláusula $(l = r \leftarrow e_1, e_2, \ldots, e_n.) \in \mathcal{E}$ tal que $l \in V$ ni f ocurre como el símbolo de

[1] En lo que sigue asumimos que las interpretaciones obedecen los axiomas de la igualdad. Consecuentemente, los conceptos de satisfacibilidad y consecuencia lógica están definidos con respecto a estos axiomas

función más externo en l; en cualquier otro caso f es un símbolo de función definido. En teorías donde se hace la distinción anterior, la signatura Σ queda particionada como $\Sigma = C \uplus F$, donde C es el conjunto de símbolos de función irreducibles y F es el conjunto de símbolos de función definidos. A continuación revisamos las nociones básicas de CLP [14].

Sea $\Pi = \Pi_C \cup \Pi_B$, con $\Pi_C \cap \Pi_B = \emptyset$. Un (Π, Σ)-*programa* es un conjunto de cláusulas de la forma: $H \leftarrow c\Box$. o $H \leftarrow c\Box B_1, \ldots, B_n$., donde c es una (Π_C, Σ)-*restricción* finita (posiblemente vacía), y H (la cabeza) y $B_1, \ldots, B_n$ (el cuerpo), $n \geq 0$, son (Π_B, Σ)-átomos. Un *objetivo* es una cláusula de programa sin cabeza.

Dada una estructura $\mathcal{X}$ [14], se dice que una restricción c es $\mathcal{X}$-*soluble* (en símbolos $\mathcal{X} \models \tilde{\exists} c$) sii existe una $\mathcal{X}$-valuación θ (una valoración sobre el soporte de $\mathcal{X}$) tal que $\mathcal{X} \models c\theta$. Se dice entonces que θ es una $\mathcal{X}$-solución de c. También se dice que c es *consistente*.

La semántica declarativa de un programa CLP puede verse tanto en términos de consecuencia lógica como con un estilo algebraico. La respuesta a un objetivo G no es una sustitución, sino una conjunción de restricciones c tal que: $(P, \mathcal{T}) \models (\forall)\ (c \Rightarrow G)$ (versión lógica) y $P \models_{\mathcal{X}} (\forall)\ (c \Rightarrow G)$ (versión algebraica), donde P es un programa CLP, $\mathcal{X}$ la estructura y $\mathcal{T}$ es una teoría que axiomatiza $\mathcal{X}$ [14]. La teoría CLP impone restricciones sobre $\mathcal{X}$, $\mathcal{T}$ y sus relaciones para establecer equivalencias entre las semánticas.

Sea P un programa CLP($\mathcal{X}$) y $G = \leftarrow c\Box A_1, \ldots, A_n$. un objetivo. Definimos una configuración como el par $\mathcal{C} = \langle \leftarrow c \Box A_1, \ldots, A_n \rangle$. Sea $G_0 = \leftarrow c_0 \Box \tilde{B}$. un objetivo inicial. La configuración $\mathcal{C}_0 = \langle \leftarrow c_0 \Box \tilde{B} \rangle$ es una configuración inicial. Una configuración terminal $\mathcal{C}$ tiene la forma $\mathcal{C} = \langle \leftarrow c \Box \ \rangle$, donde c representa la restricción computada correspondiente a esa derivación.

Un intérprete (ansioso) de un lenguaje CLP($\mathcal{X}$) se basa en una generalización de la regla de SLD-resolución utilizada en programación lógica convencional y en un algoritmo apropiado para resolver restricciones en el dominio de computación correspondiente. Intuitivamente, una computación CLP (ansiosa) puede verse como una secuencia de pasos de reducción de objetivos en la que se acumulan restricciones, posiblemente simplificadas (previa comprobación de su satisfacibilidad). La computación termina con un conjunto satisfacible de restricciones como respuesta[2].

La regla estándard que describe un paso de $(P, \mathcal{X})$-computación se define del siguiente modo [1,2,14]:

$$\frac{\exists\ n \textit{ variantes de cláusulas en } P\ (H_j \leftarrow c'_j \Box \tilde{B}_j., j = 1, \ldots, n) \wedge \tilde{c} = \{c'_1, c'_2, \ldots, c'_n, A_1 = H_1, \ldots, A_n = H_n\} \ \wedge \ \mathcal{X} \models \tilde{\exists}(c_i \cup \tilde{c})}{\langle \leftarrow c_i \Box A_1, \ldots, A_n \rangle \ \rightarrow_{CLP(\mathcal{X})} \ \langle \leftarrow c_i \cup \tilde{c} \ \Box \tilde{B}_1, \ldots, \tilde{B}_n \rangle}$$

Cuando la condición $\mathcal{X} \models \tilde{\exists}(c_i \cup \tilde{c})$ se elimina de la premisa de la regla se obtiene un intérprete perezoso para el lenguaje CLP comparable al que se describe en [11]. En [11] se asume además que un procedimiento de simplificación de restricciones se aplica en cada paso. Este procedimiento es capaz de detectar, en ocasiones, la insatisfacibilidad de la restricción.

3 Semántica Operacional (perezosa) para CLP($\mathcal{H}/\mathcal{E}$)

CLP($\mathcal{H}/\mathcal{E}$) es un lenguaje diseñado para realizar la integración de la programación lógica y ecuacional en el marco de la programación lógica con restricciones. Informalmente, un programa CLP($\mathcal{H}/\mathcal{E}$) consta de dos conjuntos de cláusulas de Horn definidas. Uno de ellos, la parte relacional, contiene sólo cláusulas cuya cabeza es un átomo (no ecuacional) y cuyo cuerpo puede contener tanto átomos como restricciones (ecuacionales). La parte funcional contiene sólamente cláusulas constituidas enteramente por ecuaciones (cláusulas de Horn ecuacionales o, equivalentemente, ecuaciones condicionales). Formalmente, sea $\Pi_C = \{=\}$, $\Pi = \Pi_C \cup \Pi_B$ y $\Pi_C \cap \Pi_B = \emptyset$. Definimos un (Π, Σ)-*programa* CLP($\mathcal{H}/\mathcal{E}$) como un (Π, Σ)-programa extendido con una Σ-teoría ecuacional de Horn *canónica* $\mathcal{E}$.

3.1 Derivaciones perezosas

En este apartado se formaliza un mecanismo computacional perezoso como semántica operacional del lenguaje CLP($\mathcal{H}/\mathcal{E}$). Nuestra noción de *computación perezosa* es similar a la noción de resolución perezosa presentada en [12] y a la noción de reducción de objetivos que se define en [11], donde la solubilidad de las restricciones no se exige en cada paso. Es evidente, no obstante, que una estrategia "demasiado" perezosa detectaría la inconsistencia de las restricciones "demasiado" tarde, produciendo una explosión inaceptable

[2] Debido a que las restricciones no se resuelven completamente en cada paso de computación se debe notar que, en el caso particular de la estructura $\mathcal{H}/\mathcal{E}$, esta forma de resolución evita el problema de la posible infinitud del número de resolventes entre una cláusula de programa y una cláusula objetivo que se presenta en las definiciones clásicas de resolución ecuacional (donde, a causa de la posible infinitud del conjunto de $\mathcal{E}$-unificadores de un par de términos, el espacio de búsqueda de la resolución puede no sólo ser infinito en profundidad sino también en amplitud) [12].

del árbol de derivaciones CLP. La capacidad para detectar la insatisfacibilidad de las restricciones es muy importante para podar caminos superfluos del árbol y evitar así computaciones innecesarias. En lo que sigue formalizamos la idea de un simplificador (incremental) de restricciones que también analiza la insatisfacibilidad de las mismas. En este contexto incrementalidad significa que no es necesario repetir todo el trabajo realizado en el paso de computación previo cuando se añade incrementalmente una nueva restricción.

DEFINICIÓN **3.1.1** (relación de transición $\longrightarrow_{LCLP(\mathcal{H}/\mathcal{E})}$). *Sea c una restricción. Supongamos que c puede escribirse de la forma* $c_1 \cup c_2$ *(cualquiera de las cuales puede ser vacía) donde* c_1 *es equivalente a un conjunto de ecuaciones en forma resuelta* $\hat{\theta}_1$ *[18]. Entonces llamamos* representación de c *al par* $\langle\, \hat{\theta}_1, c_2 \,\rangle$[3]. *En lo que sigue no haremos distinción entre restricciones y su representación. Usaremos el terminal* fail *para representar restricciones inconsistentes. La regla que describe un paso de computación perezosa para CLP*$(\mathcal{H}/\mathcal{E})$ *a partir de una configuración* $\langle \leftarrow c_i \,\square\, A_1, \ldots, A_n \rangle$ *viene dada por:*

$$\frac{\begin{array}{l}\exists\ n \textit{ variantes de cláusulas en } P\ (H_j \leftarrow c'_j \square \tilde{B}_j ., j = 1, \ldots, n) \ \wedge \\ \tilde{c} = \{c'_1, c'_2, \ldots, c'_n, A_1 = H_1, \ldots, A_n = H_n\} \ \wedge \\ c_{i+1} = \mathit{Simplificación}(c_i, \tilde{c}) \ \wedge \ c_{i+1} \neq \mathit{fail}\end{array}}{\langle \leftarrow c_i \square A_1, \ldots, A_n \rangle \ \longrightarrow_{LCLP(\mathcal{H}/\mathcal{E})} \ \langle \leftarrow c_{i+1} \square \tilde{B}_1, \ldots, \tilde{B}_n \rangle}$$

donde la función $\mathit{Simplificación}(c_i, \tilde{c})$ en la premisa de la regla reduce la restrición $c_i \cup \tilde{c}$ a una forma (más simple) equivalente mediante un conjunto apropiado de transformaciones, que se describirá posteriormente, sin comprobar su solubilidad en la estructura $\mathcal{H}/\mathcal{E}$, i.e. sin estudiar el problema de la $\mathcal{E}$-unificabilidad de las ecuaciones. El efecto de la simplificación es reducir el espacio de búsqueda de las restricciones disminuyendo, consecuentemente, el indeterminismo. La restricción c_{i+1} resultante del proceso de simplificación equivale lógicamente a la restricción $c_i \cup \tilde{c}$ original en el sentido de que tiene las mismas soluciones sobre la estructura $\mathcal{H}/\mathcal{E}$. Si la vuelta de la función es *fail*, la restricción $c_i \cup \tilde{c}$ es inconsistente.

Una configuración terminal $\mathcal{C}$ tiene la forma $\mathcal{C} = \langle \leftarrow c \,\square\, \rangle$. Si la restricción c es $\mathcal{E}$-unificable entonces representa la *restricción respuesta computada* correspondiente a esa derivación.

La corrección y completitud de este cálculo es consecuencia inmediata de los resultados en [11,14], que se aplican a cualquier lenguaje que puede formalizarse como una instancia del esquema.

3.2 Conjunto de Transformaciones Sintácticas

En la premisa de la regla que describe un paso de *computación perezosa* $\longrightarrow_{LCLP(\mathcal{H}/\mathcal{E})}$ se evalúa la función $\mathit{Simplificación}(c_i, \tilde{c})$, que reduce la restricción $c_i \cup \tilde{c}$ a una forma (más simple) equivalente. A continuación se describe en forma de reglas de inferencia el conjunto de transformaciones que definen esta función.

DEFINICIÓN **3.2.1** (relación de transición $\rightarrow_{TRANS}$). *Sea* θ *una sustitución. Sea E un conjunto de ecuaciones. Definimos una* TRANS*-configuración como un par* $T = \langle\, \hat{\theta}\ ,\ E \,\rangle$[4]. *Se define la relación* $\rightarrow_{TRANS}$ *entre restricciones mediante el siguiente sistema de transición.*
(1) Regla de eliminación de ecuaciones triviales:

$$\frac{x \in V}{\langle\, \hat{\theta}\ ,\ (\{x = x\} \cup E) \,\rangle \rightarrow_{TRANS} \langle\, \hat{\theta}\ ,\ E \,\rangle}$$

(2) Regla de reordenación:

$$\frac{x \in V \ \wedge \ t \notin V \ \wedge \ data(t)}{\langle\, \hat{\theta}\ ,\ (\{t = x\} \cup E) \,\rangle \rightarrow_{TRANS} \langle\, \hat{\theta}\ ,\ (\{x = t\} \cup E) \,\rangle}$$

(3) Regla de eliminación de variables:

$$\frac{x \in V \ \wedge \ data(t) \ \wedge \ x \notin \mathcal{V}(t)}{\langle\, \hat{\theta}\ ,\ (\{x = t\} \cup E) \,\rangle \rightarrow_{TRANS} \langle\, \theta.\widehat{\{X/t\}}\ ,\ E\theta \,\rangle}$$

(4) Regla de descomposición de términos:

[3] La restricción vacía se representa por $\langle\, \rangle$.
[4] Denotamos con $\hat{\theta}$ la representación ecuacional $\{x_1 = t_1, \ldots, x_n = t_n\}$ de la sustitución $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$.

$$\frac{f \in C}{\langle\, \hat{\theta}\,,\, (\{f(s_1,\ldots,s_n) = f(t_1,\ldots,t_n)\} \cup E)\,\rangle \rightarrow_{TRANS} \langle\, \hat{\theta}\,,\, (\{s_1 = t_1\} \cup \ldots \cup \{s_n = t_n\} \cup E)\,\rangle}$$

(5) Regla de reescritura de términos condicional[5] *:*

$$\frac{(\lambda = \rho \Leftarrow \tilde{e}) \in \mathcal{E} \;\wedge\; e/u = \lambda\sigma \;\wedge\; e' = e[u \leftarrow \rho\sigma] \;\wedge\; s\sigma \downarrow_{\mathcal{R}} t\sigma \;\; \forall (s = t) \in \tilde{e}}{\langle\, \hat{\theta}\,,\, (e \cup E)\,\rangle \rightarrow_{TRANS} \langle\, \hat{\theta}\,,\, (e' \cup E)\rangle}$$

(6) Reglas de fallo:

$$\frac{f\,,\, g \in C \wedge f \neq g}{\langle\, \hat{\theta}\,,\, (\{f(s_1,s_2,\ldots,s_n) = g(t_1,t_2,\ldots,t_m)\} \cup E)\,\rangle \rightarrow_{TRANS} fail}$$

$$\frac{x \in V \wedge data(t) \wedge x \in \mathcal{V}(t)}{\langle\, \hat{\theta}\,,\, (\{x = t\} \cup E)\,\rangle \rightarrow_{TRANS} fail}$$

siendo $data(t)$ la relación definida como:
$data(t)$ *si* $(t \in C) \vee (t \in V) \vee (t = f(t_1,\ldots t_2) \wedge f \in C \wedge (\forall i,\, 1 \leq i \leq n, data(t_i)))$

DEFINICIÓN **3.2.2** (TRANS-configuración terminal T). *Definimos el conjunto de* TRANS*-configuraciones terminales de la siguiente forma:* $\{\, \langle\, \hat{\theta}\,,\, E\,\rangle \,/\, \langle\, \hat{\theta}\,, E\,\rangle \not\rightarrow_{TRANS} \,\} \cup \{fail\}$.

Es conocido que cada regla del cálculo $\rightarrow_{TRANS}$ transforma un conjunto de ecuaciones $\hat{\theta} \cup E$ en otro conjunto $\hat{\theta}' \cup E'$ con el mismo conjunto de $\mathcal{H}/\mathcal{E}$-soluciones [12]. Si el cálculo termina en *fail* el conjunto de ecuaciones original es inconsistente. Claramente, el cálculo $\rightarrow_{TRANS}$ siempre termina para teorías ecuacionales canónicas y es independiente del orden en que hayan sido aplicadas las reglas[6] [12]. Se debe notar la incrementalidad en el proceso de simplificación de las restricciones gracias a la representación adoptada.

DEFINICIÓN **3.2.3** (Comportamiento del cálculo $\rightarrow_{TRANS}$).
Sea $T_0 = \langle\, \hat{\theta}\,,\, (c \cup \tilde{c}\theta)\,\rangle$. *Definimos el comportamiento de la relación* $\rightarrow_{TRANS}$ *como la función:*

$$Simplificación(\langle\, \hat{\theta}\,,\, c\,\rangle, \tilde{c}) = \begin{cases} \langle\, \hat{\theta}'\,,\, c'\,\rangle & si\ T_0 \rightarrow^*_{TRANS} \langle\, \hat{\theta}'\,,\, c'\,\rangle \\ fail & si\ T_0 \rightarrow^*_{TRANS} fail \end{cases}$$

El siguiente ejemplo ilustra cómo el mecanismo anteriormente descrito evita el problema de la no terminación de un paso de computación CLP($\mathcal{H}/\mathcal{E}$) ansiosa cuando la restricción acumulada no es satisfacible. Se debe notar que nuestro concepto de derivación perezosa recupera la capacidad de computar funciones que no posee el intérprete CLP($\mathcal{H}/\mathcal{E}$) que se obtiene como instancia de la definición estándar presentada en la sección 2.

EJEMPLO **3.2.4**. *Sea* $(\mathcal{P}, \mathcal{E})$ *un programa CLP($\mathcal{H}/\mathcal{E}$) definido por las siguientes cláusulas:*
$(p_1)\ p(f(Z)) \leftarrow \Box\, q(Z).$ $(p_2)\ q(a).$ $(e_1)\ f(a) = a.$ $(e_2)\ f(d(X)) = f(X).$
Consideremos el objetivo inicial $G_0 = \leftarrow \Box p(b)$. *La configuración CLP($\mathcal{H}/\mathcal{E}$) inicial es* $C_0 = \langle \leftarrow \Box p(b)\rangle$. *Considerando la cláusula* (p_1), *la nueva restricción* $\tilde{c}_1$ *es* $\tilde{c}_1 = \{\ b = f(Z)\ \}$.

Un intérprete ansioso para CLP($\mathcal{H}/\mathcal{E}$) debería decidir la satisfacibilidad de la restricción $\tilde{c}_1$ *antes de efectuar el correspondiente paso de computación. El intento de $\mathcal{E}$-unificar la restrición (inconsistente)* $\tilde{c}_1$ *utilizando la regla* (e_1) *fallaría al derivar la ecuación (ground) inconsistente* $\{b = a\}$. *Sin embargo la derivación* $b = f(Z) \rightarrow_{\{Z/d(X)\}(e_2)} b = f(X) \rightarrow_{\{X/d(X')\}(e_2)} b = f(X') \rightarrow \ldots$ *es infinita por aplicación de la regla* (e_2).

Teniendo en cuenta que $Simplificación(\langle\,\rangle, \{b = f(Z)\}\,\rangle) = \langle\, \emptyset\,,\, \{b = f(Z)\}\,\rangle = c'_1$, *el mecanismo computacional perezoso descrito en la sección 3.1 puede probar la transición* $C_0 \longrightarrow_{LCLP(\mathcal{H}/\mathcal{E})} \langle c'_1 \Box\, q(Z)\,\rangle$

Considerando ahora la cláusula (p2), la nueva restricción es $\tilde{c}_2 = \{Z = a\}$. *Teniendo en cuenta que* $\langle\, \emptyset\,,\, (\{b = f(Z)\} \cup \{Z = a\})\,\rangle \rightarrow_{TRANS\ (3)} \langle\, \{Z = a\}\,,\, \{b = f(a)\}\,\rangle \rightarrow_{TRANS\ (5)} \langle\, \{Z = a\}\,,\, \{b = a\}\,\rangle$

[5] Las variables de la cláusula $(\lambda = \rho \Leftarrow \tilde{e})$ se asumen estandarizadas aparte.

[6] Para la terminación de la regla de reescritura condicional el sistema de reescritura debe ser decreciente en el sentido de [16,21]. Esta propiedad es más fuerte que la de terminación y asegura la decibilidad de la relación $\rightarrow_{\mathcal{R}}$ para sistemas de reescritura condicionales.

$\rightarrow_{TRANS\ (e^i)}$ *fail* *se tiene que* $c'_2 = Simplificación(\langle\ \emptyset\ ,\ \{b = f(Z)\}\ \rangle, \{Z = a\}) = fail$ *y, como consecuencia, no es posible probar la correspondiente transición* $\longrightarrow_{LCLP(\mathcal{H}/\mathcal{E})}$. *El objetivo* G_0 *fracasa, por tanto, finitamente.*

EJEMPLO 3.2.5. *Sea el siguiente programa CLP($\mathcal{H}/\mathcal{E}$) que define el número de elementos de una lista:*
(e_1) n_elementos(λ) = 0. *(e_2) n_elementos(X.T) = 1+n_elementos(T).*
Consideremos el objetivo inicial $G_0 = \leftarrow$ *n_elementos(a.b.c) = X* $\square$. *El intérprete (ansioso) que se describe en la sección 2 comprueba la satisfacibilidad de la restricción en el objetivo* G_0 *y devuelve la propia restricción como respuesta. El intérprete que se describe en esta sección simplifica la restricción n_elementos(a.b.c) = X computando la restricción respuesta X = 3.*

3.3 Optimización del cálculo $\rightarrow_{TRANS}$

A continuación presentamos una optimización del cálculo de simplificación de restricciones que está basado en la idea de convergencia de operadores *(operator joinability)* que se describe en [7]. En lo que sigue extendemos dicha noción al caso de sistemas de reescritura condicionales. En [3] se redefine este concepto en el marco formal de la teoría de la Interpretación Abstracta.

Sea $\mathcal{R}$ un sistema de reescritura de términos sobre la signatura Σ. Construimos un sistema de reescritura $\mathcal{R}'$ derivado de $\mathcal{R}$ de la siguiente forma:

- Para cada $(f(t_1, t_2, \ldots, t_n) \rightarrow g(s_1, s_2, \ldots, s_m) \Leftarrow \tilde{e}) \in \mathcal{R}$, se añade a $\mathcal{R}'$ la regla $\{f \rightarrow g \Leftarrow out(\tilde{e})\}$.
- Para cada $(f(t_1, t_2, \ldots, t_n) \rightarrow X \Leftarrow \tilde{e}) \in \mathcal{R}$, $X \in V$ y para cada $g_i \in \Sigma$ se añade a $\mathcal{R}'$ la regla $\{f \rightarrow g_i \Leftarrow out(\tilde{e}, X, g_i)\}$.

donde $out(\tilde{e}) = \{out(t) = out(s) \mid (t = s) \in \tilde{e}\}$ siendo $out(t)$ el símbolo más externo de t si $t \notin V$ y t en cualquier otro caso, y $out(\tilde{e}, X, g_i)$ es el (multi-)conjunto de ecuaciones $out(\tilde{e}, X, g_i) = \{out(t, X, g_i) = out(s, X, g_i) \mid (t = s) \in \tilde{e}\}$ en el que $out(t, X, g_i)$ es el símbolo de función más externo del término t si $t \notin V$, g_i si t es la variable X y t en cualquier otro caso.

La noción de convergencia de operadores es una noción decidible que puede ser implementada, e.g., mediante un sencillo proceso que construye el grafo de la relación $\rightarrow_{\mathcal{R}'}$ de acuerdo con las reglas del sistema de reescritura $\mathcal{R}'$. Dos símbolos de función $f, g \in \Sigma$ son convergentes en $\mathcal{R}'$, $f \downarrow_{\mathcal{R}'} g$, si los caminos en el grafo que parten de los vértices correspondientes tienen, al menos, un vértice común [23]. Se debe notar que el conjunto de vértices de cada grafo es finito puesto que el número de reglas en $\mathcal{R}'$ lo es.

PROPOSICIÓN 3.3.1 ([7]). *Una ecuación* $f(\tilde{t}) = g(\tilde{s})$ *es satisfacible en* $\mathcal{R}$ *sólo si* $f \downarrow_{\mathcal{R}'} g$.

El cálculo $\rightarrow_{TRANS}$ puede reforzarse haciendo uso del resultado anterior con la introducción de la siguiente regla:

$$\frac{f \not\downarrow_{\mathcal{R}'} g}{(\{f(s_1, s_2, \ldots, s_n) = g(t_1, t_2, \ldots, t_m)\} \cup E) \rightarrow_{TRANS} fail}$$

EJEMPLO 3.3.2. *Sea $(P, \mathcal{E})$ un programa CLP($\mathcal{H}/\mathcal{E}$) definido por las siguientes reglas:*
(p_1) p(b(Z)) $\leftarrow$ $\square$ q(Z). *(p_2) q(c).* *(e_1) a(f(X)) = a(X).* *(e_2) b(f(X)) = b(X).*
Sea $G_0 = \leftarrow \square\ p(a(X))$ *el objetivo inicial. La configuración inicial es entonces* $C_0 = \langle \leftarrow \square\ p(a(X))\ \rangle$. *Utilizando la regla ($p_1$) del programa se tiene la restricción acumulada* $\tilde{c} = \{\ a(X) = b(Z)\ \}$. *La inconsistencia de $\tilde{c}$ sólo puede reconocerse tras detectar la no convergencia de los operadores más externos a y b, i.e. a* $\not\downarrow_{\mathcal{R}}$ *b, respecto a las reglas del sistema derivado* $\mathcal{R}' = \{\ a \rightarrow_{\mathcal{R}'} a,\ b \rightarrow_{\mathcal{R}'} b\ \}$. *El cálculo* $\rightarrow_{TRANS}$ *original no detecta esta inconsistencia dado que ninguna de las reglas de transformación es aplicable a $\tilde{c}$.*

Referencias bibliográficas

[1] M. Alpuente and M. Falaschi. Incremental Constraint Satisfaction for Equational Logic Programming. In J. Maluszynski and M. Wirsing, editors, *Proc. of PLILP'91*, volume 528 of *Lecture Notes in Computer Science*, pages 111–122. Springer-Verlag, Berlin, 1991.

[2] M. Alpuente, M. Falaschi, and G. Levi. Incremental Constraint Satisfaction for Equational Logic Programming. Technical Report TR 30/91, Dipartimento di Informatica, Università di Pisa, 1991. submitted for publication. Extended abstract in *Proc. ILPS'91 Workshop on Defeasible Reasoning and Constraint Solving*, pages 47-75. San Diego, Ca., Oct.1991.

[3] M. Alpuente, M. Falaschi, and F. Manzo. Analyses of Inconsistency for Incremental Equational Logic Programming. In *Proc. of PLILP'92*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1992.

[4] M. Alpuente and M.J. Ramírez. An Equational Constraint Logic Approach to Conceptual Modelling. In *Proc. of DEXA'92*. Springer-Verlag, Berlin, 1992.

[5] M. Bellia and G. Levi. The relation between logic and functional languages: a survey. *Journal of Logic Programming*, 3:217–236, 1986.

[6] P. Bosco, E. Giovannetti, and C. Moiso. Narrowing vs. SLD-resolution. *Theoretical Computer Science*, 59:3–23, 1988.

[7] N. Dershowitz and J. Sivakumar. Solving Goals in Equational Languages. In S. Kaplan and J.Joaunnaud, editors, *Proc. First Int'l Workshop on Conditional Term Rewriting*, volume 308 of *Lecture Notes in Computer Science*, pages 45–55. Springer-Verlag, Berlin, 1987.

[8] L. Fribourg. Slog: a logic programming language interpreter based on clausal superposition and rewriting. In *Proc. Second IEEE Int'l Symp. on Logic Programming*, pages 172–185. IEEE, 1985.

[9] J.H. Gallier and W. Snyder. A general complete E-unification procedure. In P. Lescanne, editor, *Rewriting Techniques and Applications*, volume 256 of *Lecture Notes in Computer Science*, pages 216–227. Springer-Verlag, Berlin, 1987.

[10] E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi. Kernel Leaf: A Logic plus Functional Language. *Journal of Computer and System Sciences*, 42, 1991.

[11] M. Höhfeld and G. Smolka. Definite relations over constraint languages. Technical report, IBM Deutschland GmbH, Stuttgart, 1988.

[12] S. Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 1989.

[13] H. Hussman. Unification in conditional-equational theories. Technical report, Fakultät für Mathematik und Informatik, Universität Passau, 1986.

[14] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *Proc. Fourteenth Annual ACM Symp. on Principles of Programming Languages*, pages 111–119. ACM, 1987.

[15] J. Jaffar, J.-L. Lassez, and M.J. Maher. A logic programming language scheme. In D. de Groot and G. Lindstrom, editors, *Logic Programming, Functions, Relations and Equations*, pages 441–468. Prentice Hall, Englewood Cliffs, NJ, 1986.

[16] S. Kaplan. Fair conditional term rewriting systems: unification, termination and confluence. In H.-J. Kreowski, editor, *Recent Trends in Data Type Specification*, volume 116 of *Informatik-Fachberichte*, pages 136–155. Springer-Verlag, Berlin, 1986.

[17] J.W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, I. Oxford University Press, 1991.

[18] J.-L. Lassez, M.J. Maher, and K. Marriott. Unification Revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann, Ca., 1988.

[19] J.W. Lloyd. *Foundations of logic programming*. Springer-Verlag, Berlin, 1987. second edition.

[20] A. Martelli, C. Moiso, and G.F. Rossi. An algorithm for unification in equational theories. In *Proc. IEEE Symp. on Logic Programming*, pages 180–186. IEEE, 1986.

[21] A. Middeldorp and E. Hamoen. Counterexamples to completeness results for basic narrowing. To appear in Proc. Third Int'l Conf. on Algebraic and Logic Programming, 1992.

[22] P. Padawitz. *Computing in Horn Clause Theories*, volume 16 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1988.

[23] M.J. Ramis and M. Alpuente. Un intérprete perezoso para CLP($\mathcal{H}/\mathcal{E}$). Technical report, Universidad Politécnica de Valencia, Valencia, Spain, 1992. forthcoming.

[24] J.H. Siekmann. Universal unification. In *7th Int'l Conf. on Automated Deduction*, volume 170 of *Lecture Notes in Computer Science*, pages 1–42. Springer-Verlag, Berlin, 1984.

[25] H. Tamaki. Semantics of a Logic Programming Language with a Reducibility Predicate. In *Proc. First IEEE Int'l Symp. on Logic Programming*, pages 259–264. IEEE, 1984.

Anexo. Un intérprete Prolog (perezoso) para CLP($\mathcal{H}/\mathcal{E}$)

Un intérprete Prolog (perezoso) para programas CLP($\mathcal{H}/\mathcal{E}$) se define, esencialmente, mediante cuatro predicados: `solve/3`, `ics/3`, `simplify/5` e `inwing/3`. El primero de estos predicados conduce las transiciones del nivel CLP($\mathcal{H}/\mathcal{E}$) del intérprete. Los otros dos realizan el segundo nivel del sistema, simplificando (incrementalmente) las restricciones y detectando, en su caso, su insatisfacibilidad. El

predicado `inwing/3` define un procedimiento de narrowing condicional básico que se puede utilizar para comprobar la satisfacibilidad de las restricciones en cada configuración terminal.

En la definición del modelo operacional del lenguaje se adoptó la regla de computación estándar del modelo CLP, en la que todos los átomos existentes en el objetivo se seleccionan simultáneamente [14]. En la implementación se utilizan las reglas estándar *left-to-right* y *top-to-bottom* en profundidad de Prolog. A continuación describimos el procedimiento `solve/3` que implementa el nivel superior del intérprete CLP($\mathcal{H}/\mathcal{E}$).

Las cláusulas de programa se almacenan como cláusulas unit: `clause_(Head,Body,ConstrBody)` correspondientes a una regla del programa `Head:- ConstrBody` □ `Body.`, donde `ConstrBody` y `Body` son, respectivamente, la parte ecuacional y relacional del cuerpo de la cláusula. Los argumentos del predicado `solve/3` son: la lista de átomos a resolver, el estado de partida del simplificador de restricciones y el estado con que termina.

```
solve([],ICS_state,ICS_state).
solve([Goal|Restgoal],PrevICS_state,NewICS_state):-
    solve(Goal,PrevICS_state,TempICS_state),
    solve(Restgoal,TempICS_state,NewICS_state).
solve(Goal,PrevICS_state,NewICS_state):-
    flatgoal(Goal,Flatgoal,Listeqs), clause_(Flatgoal,Body,ConstrBody),
    append(Listeqs,ConstrBody,TildeConstr),
    ics(PrevICS_state,TildeConstr,TempICS_state),
    solve(Body,TempICS_state,NewICS_state).
```

El procedimiento `flatgoal/3` realiza un aplanamiento del subobjetivo `Goal`, sustituyendo cada uno de sus argumentos por una nueva variable y añadiendo la ecuación que expresa la igualdad entre ambos a la lista de restricciones `Listeqs`. El núcleo del procedimiento `solve/3` es el predicado `ics/3`, que define el segundo nivel del sistema. Los argumentos de `ics/3` son: el estado de partida, la nueva restricción $\tilde{c}$ a acumular y el estado resultante que incorpora la restricción (simplificada) acumulada. El predicado `simplify/5` define el nivel inferior del sistema, implementando las reglas del sistema $\rightarrow_{TRANS}$ descritas en la subsección 3.2.

```
ics(PrevICS_state,Tildeconstr,NewICS_state):-
    PrevICS_state=(Constr,Hat_theta), NewICS_state=(Constr_prima,Hat_theta_prima),
    apply_sust(Tildeconstr,Hat_theta,Tilde_hat_constr), append(Constr,Tilde_hat_constr,Tempconstr),
    simplify(Hat_theta,Tempconstr,Hat_theta_prima,Constr_prima,Fail), Fail.
```

Las cláusulas de la teoría ecuacional están almacenadas como cláusulas Prolog con la siguiente forma `=-(Left,Right):- Condition.` Siguiendo las ideas de [6], se ha programado un procedimiento `init/0` que computa el conjunto de ocurrencias no variables de las partes derechas de las cabezas de las cláusulas de la teoría ecuacional:

```
init:- clause(=-(L,R),C), occ(R,OR), assertz(rwr(L,R,OR,C)), fail
init.
occ(T,O) :- var(T), !.
occ(T,[1|Ann]):- T=..[F|ARG], occ1(ARG,Ann).
occ1([],[]).
occ1([TH|TT],[A|Ann]):- occ(TH,A), occ1(TT,Ann).
```

El procedimiento `inwing/2` implementa un procedimiento de narrowing condicional con regla de computación *left-to-right* y regla de búsqueda *top-to-bottom* en profundidad.

```
inwing(T,S) :- occ(T,OT), occ(S,OS), !, narred(T,OT,TR), narred(S,OS,SR), TR=SR.
narred(T,O,T):- !.
narred(TC,[1|Oarg],TR):- TC=..[F|Arg], narrarg(Arg,Oarg,Narg),
    TC1=..[F|Narg], narrterm(TC1,TR).
narrterm(T,T).
narrterm(TC,TR):- rwr(TC,R,OR,T), call(T), narred(R,OR,T).
narrarg([],[],[]).
narrarg([A1|Arg],[O1|Oarg],[AR|ArggR]):- narred(A1,O1,AR), narrarg(Arg,Oarg,ArgR).
```

Herramienta Gráfica Interactiva para Métodos de Elementos Finitos en 3 Dimensiones

María Cecilia Rivara
Depto. de Ciencias de la Computación
Universidad de Chile
Casilla 2777, Santiago, Chile
E-mail: mcrivara@dcc.uchile.cl

Resumen

En este trabajo se describe una herramienta gráfica de gran utilidad en el análisis de problemas tridimensionales mediante el método de elementos finitos. Esta permite la modificación interactiva de mallas de tetraedros en estaciones de trabajo con ambiente multiventanas.

Con el fin de interactuar con la malla de tetraedros, el usuario dispone de tres vistas planas ortográficas del objeto y una vista tridimensional paralela oblicua de éste, sobre las cuales manipula y superpone a voluntad un patrón de refinamiento. Una vez ubicado y dimensionado adecuadamente el patrón de refinamiento, el usuario utiliza un algoritmo versátil y confiable para refinar todos los tetraedros que intersectan el patrón de refinamiento manteniendo la suavidad y buenas propiedades geométricas de las mallas.

1 Introducción

Un área de aplicación de gran impacto de las técnicas CAD es el análisis de problemas de ingeniería mediante el uso de métodos de elementos finitos. Aunque estas técnicas se han usado extensamente en las últimos 15 años, aun hoy se asume que un ser humano tomará las decisiones y analizará los resultados de cada etapa del proceso [3].

Sin embargo, desde que Babuska y Rheinboldt describieron en 1977 [1] las características deseables de los sistemas de elementos finitos totalmente automáticos, la aplicación de técnicas adaptivas en esta área ha suscitado gran interés. En [5], la autora de este artículo ha discutido en detalle, las características del software adaptivo de elementos finitos que permitiría explotar adecuadamente el potencial del método de elementos finitos, reduciendo tanto el riesgo de mala utilización de éste, como el trabajo humano involucrado en la definición de la discretización.

El diseño de software adaptivo se basa en la construcción de la malla en forma dinámica a través del proceso. Esto depende en forma crítica de la disponibilidad de algoritmos flexibles que permitan modificar la malla, ya sea por refinamiento y/o por desrefinamiento, si el error estimado sobre la discretización en curso no es aceptable. Este trabajo se realiza en forma automática u adaptiva si el software incluye estimadores confiables del error asociados a la ecuación diferencial en consideración.

Sin embargo, a pesar del extenso trabajo realizado en esta dirección en los últimos 10 años, la mayor parte del trabajo adaptivo ha sido de interés académico y realizado dentro de las universidades. Poco se ha incluído en paquetes comerciales [3]. Menos trabajo aún se ha hecho en este sentido en tres dimensiones.

La misma estructura de resolución puede utilizarse en forma interactiva si el usuario dispone de herramientas gráficas y algoritmos que le permitan fácilmente comparar las soluciones obtenidas sobre dos mallas distintas, donde la malla más fina se ha obtenido por refinamiento de la otra. El usuario dispondría así de una herramienta que le permitiera asegurar que el análisis realizado tiene un grado aceptable de confiabilidad.

En este trabajo se discute una herramienta que combina un algoritmo de refinamiento confiable y versátil para malla de tetraedros y un criterio interactivo basado en el concepto de patrón de refinamiento controlado por el usuario.

2 Diseño Básico del Software Adaptivo o Interactivo

El diseño de software adaptivo/interactivo para análisis de elementos finitos, se centra en el manejo y modificación de la malla a través del proceso, siguiendo el siguiente esquema general

0. Construcción de la malla inicial.
1. Cálculo de la solución de elementos finitos.
2. Evaluación de la malla (solución de elementos finitos).
3. Mientras la malla no es aceptable, se realiza lo siguiente:
   - 3.1. Definición de la submalla de refinamiento
   - 3.2. Refinamiento de la malla
   - 3.3. Cálculo de la solución de elementos finitos.
   - 3.4. Evaluación de la malla (solución de elementos finitos).

La solución numérica obtenida en cada paso depende de la calidad de la malla construída para discretizar el problema continuo. La evaluación de la malla se realiza en la práctica analizando dicha solución numérica.

En el contexto adaptivo, el proceso completo se realiza en forma totalmente automática usando un estimador confiable del error en los pasos 2 y 3.4 y un criterio que permita decidir qué elementos deberían refinarse para obtener una mejor solución (submalla de refinamiento) [1,5].

En el contexto interactivo en cambio, los pasos 2 y 3.4 dependen del usuario y de las herramientas de que disponga para ayudarlo a definir la submalla de refinamiento.

En este trabajo se describe una herramienta gráfica que permite el manejo interactivo de mallas de tetraedros, que puede integrarse fácilmente con técnicas adaptivas en una etapa posterior. Esta se basa en la combinación de un algoritmo de refinamiento en 3D basado en la bisección de tetraedros por el lado más largo [7] (que ha mostrado ser muy efectivo en la práctica), y el concepto de patrón de refinamiento [4].

El patrón de refinamiento es una primitiva gráfica tridimensional, que el usuario puede manipular en ubicación y en tamaño, de modo tal que su posición final con respecto al sólido, define la submalla de refinamiento. La submalla de refinamiento está constituída así por el conjunto de tetraedros de la malla que intersectan el patrón de refinamiento.

El algoritmo de refinamiento trabaja en forma local sobre esta submalla de refinamiento y asegura que la malla final construída es conforme (la intersección de tetraedros vecinos es un vértice, una arista o una cara común).

Cada una de las mallas generadas en el proceso, puede usarse a su vez como malla de entrada para continuar con nuevos refinamientos.

3 El Algoritmo de Refinamiento

En [7] se discute el algoritmo en tres dimensiones que hemos utilizado para modificar por refinamiento mallas de tetraedros, basado en las siguientes ideas y definiciones.

Cualquier tetraedro no degenerado T se define por sus 4 vértices no coplanares. La bisección generalizada de T es la división de éste en dos tetraedros mediante un plano determinado por el punto medio de la arista más larga de T y los dos vértices restantes.

Una malla de tetraedros τ es conforme si la intersección de dos tetraedros vecinos es un vértice, arista o cara común. Observe que τ es una malla no conforme si en general existe al menos un punto no conforme P que es simultáneamente punto interior del lado de al menos un tetraedro y vértice de al menos dos tetraedros.

Sea τ una malla conforme de tetraedros y T un tetraedro a ser refinado localmente en τ. El algoritmo que hemos utilizado realiza el mínimo número de bisecciones generalizadas recursivas sobre T y sus vecinos para obtener una malla conforme.

Para describir el algoritmo en forma precisa y simple definiremos el conjunto vecino de un tetraedro T como el conjunto de tetraedros adyacentes que comparten con T su lado más largo. Esta definición nos permite identificar todas los tetraedros que deberían refinarse (para asegurar conformidad) si se realiza una bisección generalizada de T.

El algoritmo se describe así mediante el siguiente procedimiento recursivo:

Refinamiento-de-tetraedro (T, τ)
Realice una bisección generalizada de T
(Sea P el nuevo vértice generado)
Mientras P es no conforme realice lo siguiente
Encuentre el conjunto vecino N_T de T
para cada $T^* \in N_0$ realice lo siguiente
Refinamiento-de-tetraedro (T^*, τ)

Aunque no se ha demostrado matemáticamente que el algoritmo anterior preserva la forma de los elementos, en el sentido que los elementos no degeneran cuando el proceso prosigue hasta el infinito, como se ha demostrado para su análogo bidimensional [6], en [7] se ha incluído experimentación numérica que muestra que el algoritmo $3D$ tiene en la práctica el mismo tipo de comportamiento.

4 Patrón de Refinamiento y Refinamiento Interactivo

Con el fin de obtener una herramienta interactiva efectiva para refinamiento de mallas cualesquiera en $3D$, hemos combinado técnicas de visualización de CAD y la idea de patrón de refinamiento.

El patrón de refinamiento es un sólido standard (en este caso una caja) que el usuario puede mover, agrandar o reducir utilizando un dispositivo bidimensional (mouse).

Una vez que el usuario decide realizar refinamientop interactivo, un menú le permite desplegar cuatro vistas de la malla en curso: tres proyecciones planas ortográficas y una vista tridimensional paralela oblicua. Sobre estas vistas se superponen las vistas correspondientes al patrón de refinamiento. Mediante estas vistas y el "mouse", el usuario puede manipular y ubicar el patrón de refinamiento, de modo que su dimensión y posición final define la submalla de refinamiento como el conjunto de tetraedros que lo intersectan.

Las figuras 1 y 2 ilustran las ideas anteriores para una malla gruesa inicial. la Figura 1 muestra la posición y tamaño del patrón de refinamiento inicial. En la Figura 2 éste ha sido manipulado y posicionado sobre la malla, definiendo una submalla de refinamiento. La Figura 3 muestra una vista tridimensional de la malla generada al aplicar una iteración del algoritmo de refinamiento.

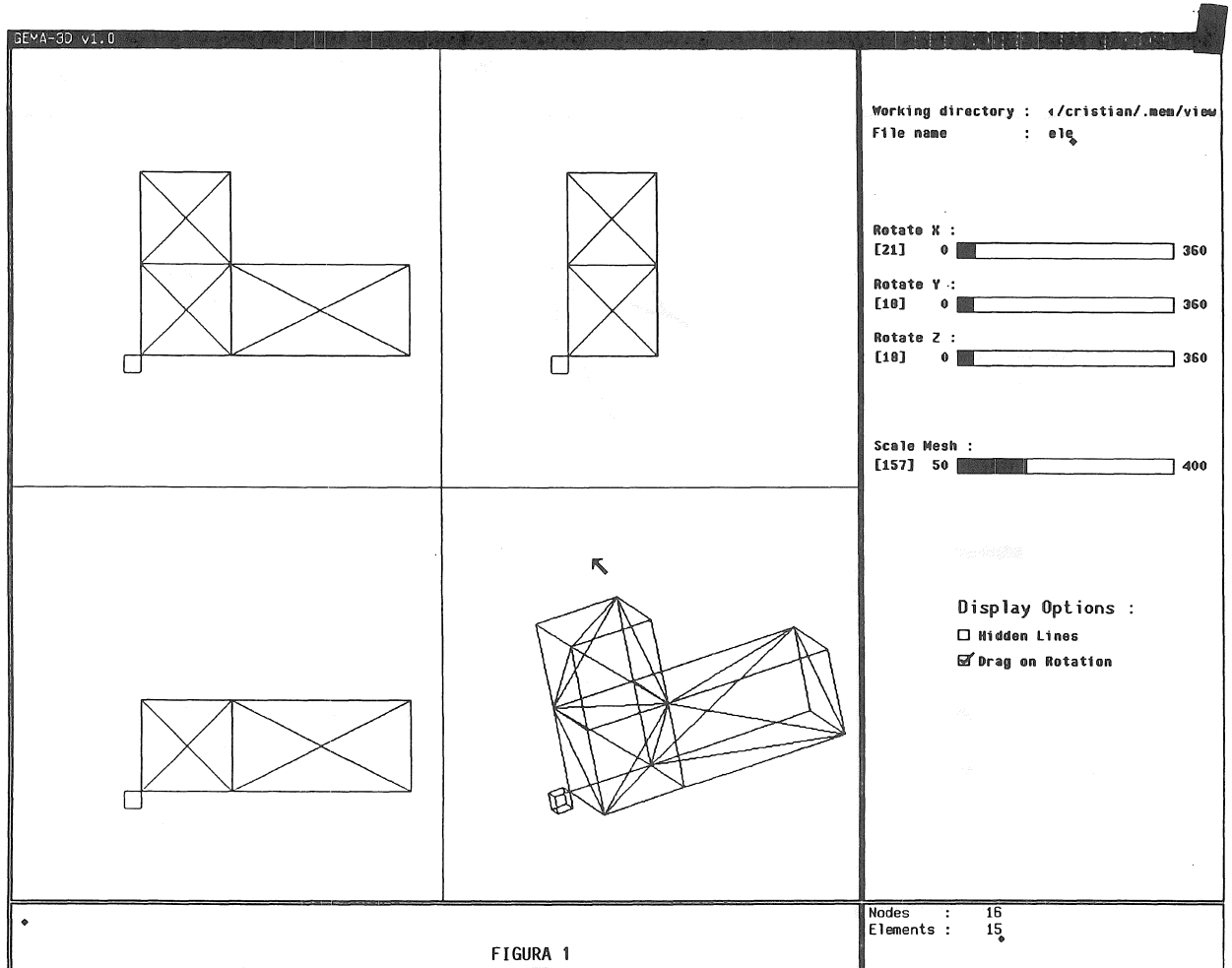
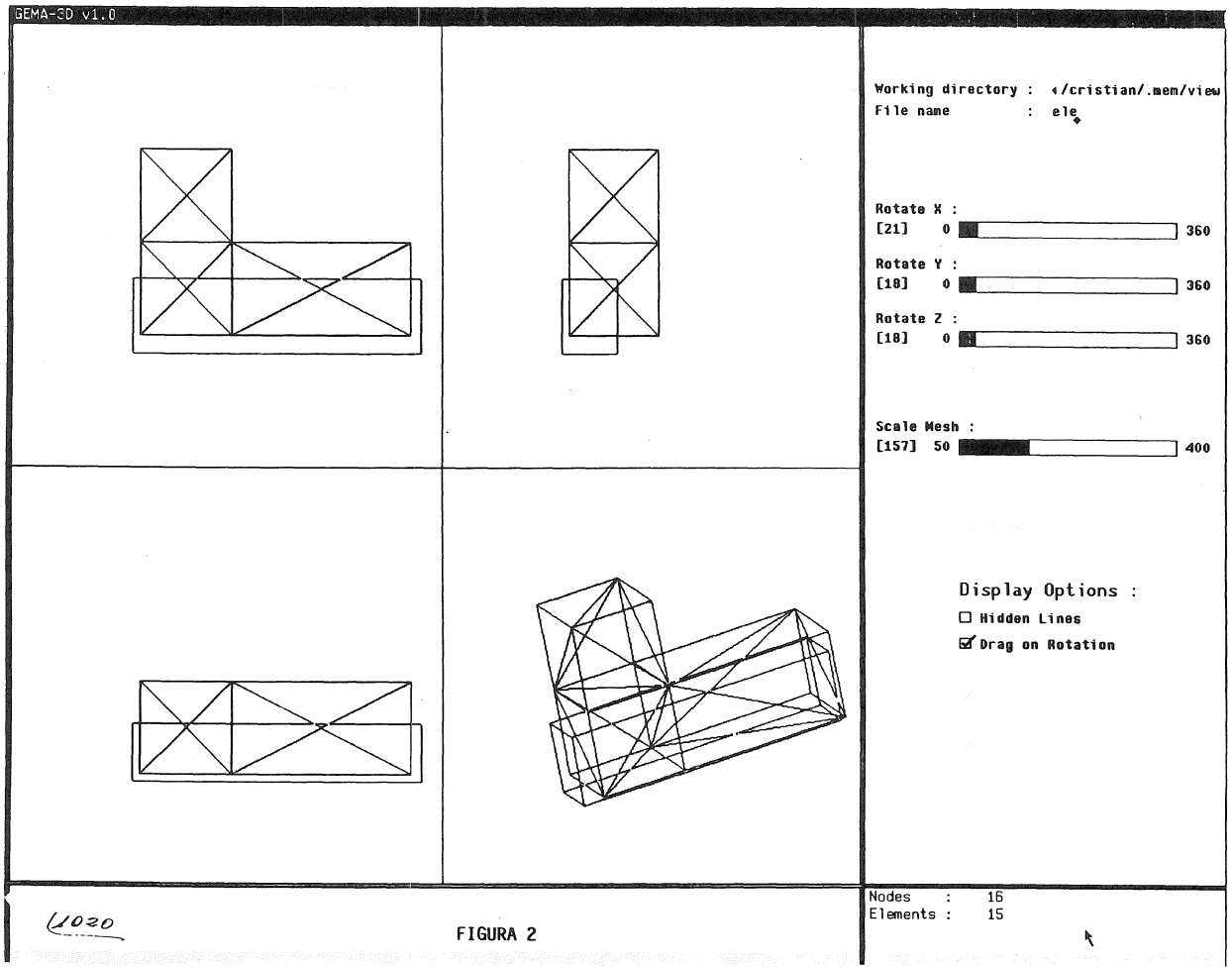

GEMA-3D v1.0
Working directory : ‹/cristian/.mem/view
File name : ele
Rotate X :
[21] 0 360
Rotate Y :
[10] 0 360
Rotate Z :
[10] 0 360
Scale Mesh :
[157] 50 400
Display Options :
Hidden Lines
Drag on Rotation
Nodes : 16
Elements : 15

FIGURA 1



1020

FIGURA 2

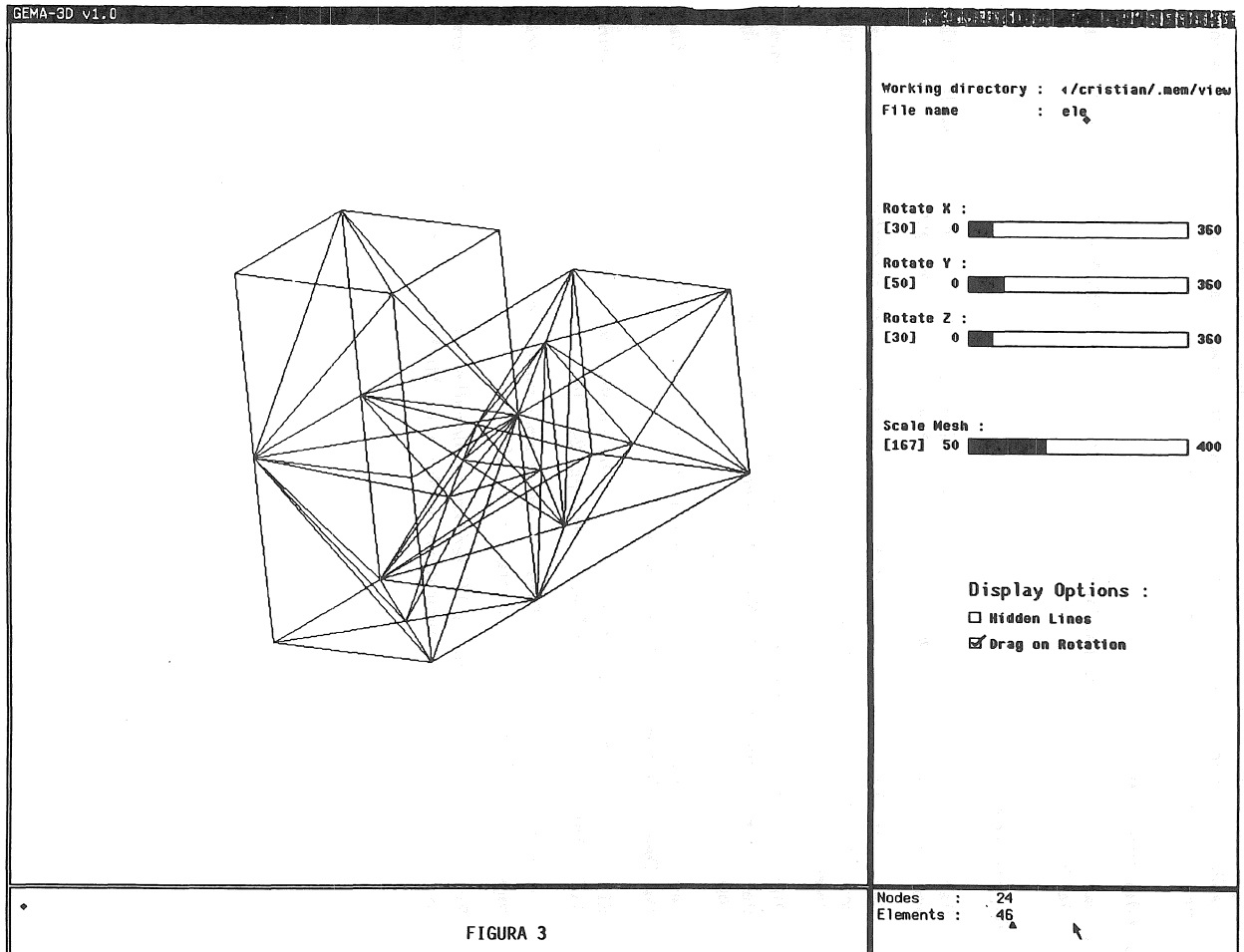

GEMA-3D v1.0
Working directory : ‹/cristian/.mem/view
File name : ele
Rotate X :
[30] 0 360
Rotate Y :
[50] 0 360
Rotate Z :
[30] 0 360
Scale Mesh :
[167] 50 400
Display Options :
Hidden Lines
Drag on Rotation
Nodes : 24
Elements : 46

FIGURA 3

La Figura 4 muestra una malla generada después de 9 pasos de refinamiento iterativo usando el mismo patrón de refinamiento indicado en la figura. La Figura 5 muestra un conjunto de estadísticas calculadas para cada una de las mallas construídas durante las 9 iteraciones. Por ejemplo, para la malla de la Figura 5 los ángulos planos asociados al peor ángulo sólido generado son 35.26°, 25.35° and 18.43°. Además esta malla tiene sólo un 2.65% de elementos cuyo ángulo sólido mínimo es menor que 10°. Observe también que a partir de la iteración 4 se ha mantenido constante el peor ángulo sólido construído. Más informaciones acerca de estas estadísticas y sobre la experimentación numérica realizada con estos algoritmos puede encontrarse en [7].

Agradecimientos

Este trabajo ha sido financiado parcialmente por el Proyecto FONDECYT 91-1220 y el Proyecto DTI 2847/902F.

Referencias

1. Babuska, I. and W.C. Rheinboldt. *Computational aspects of finite element method.* In Mathematical Software III. J.R. Rice, Ed. Academic Press, 1977, pp. 225-255.

2. Cavendish, J.C., D.A. Field and W.H. Frey. *Automatic mesh generation: a finite element computer aided geometric design interface.* MAFELAP 1984, J.R. Whiteman(Ed.), Academic Press, 1985, pp. 83-96.

3. Hinton, E., M.A. Crisfield and D. Hitchings. *Research group seminar on adaptive meshing.* Benchmark, Newsletter of the National Agency for Finite Element Methods and Standards, 1990, pp. 9-12.

4. Levin, C., *Generación interactiva de mallas tridimensionales para métodos de elementos finitos.* Depto. Ciencias de la Computación, 1991.

5. Rivara, M.C. *Design and data structure of fully adaptive, multigrid, finite-element software.* ACM Trans. Math. Software Vol.10, 1984, pp. 242-264.

6. Rivara, M.C. *Algorithms for refining triangular grids suitable for adaptive and multigrid techniques.* Int. J. Numer. Meth. Engrg., Vol.20, 1984, pp. 745-765.

7. Rivara, M.C. and C. Levin. *A 3D refinement algorithm suitable for adaptive and multigrid techniques.* To appear Communications on Numerical Methods in Engineering, 1992.

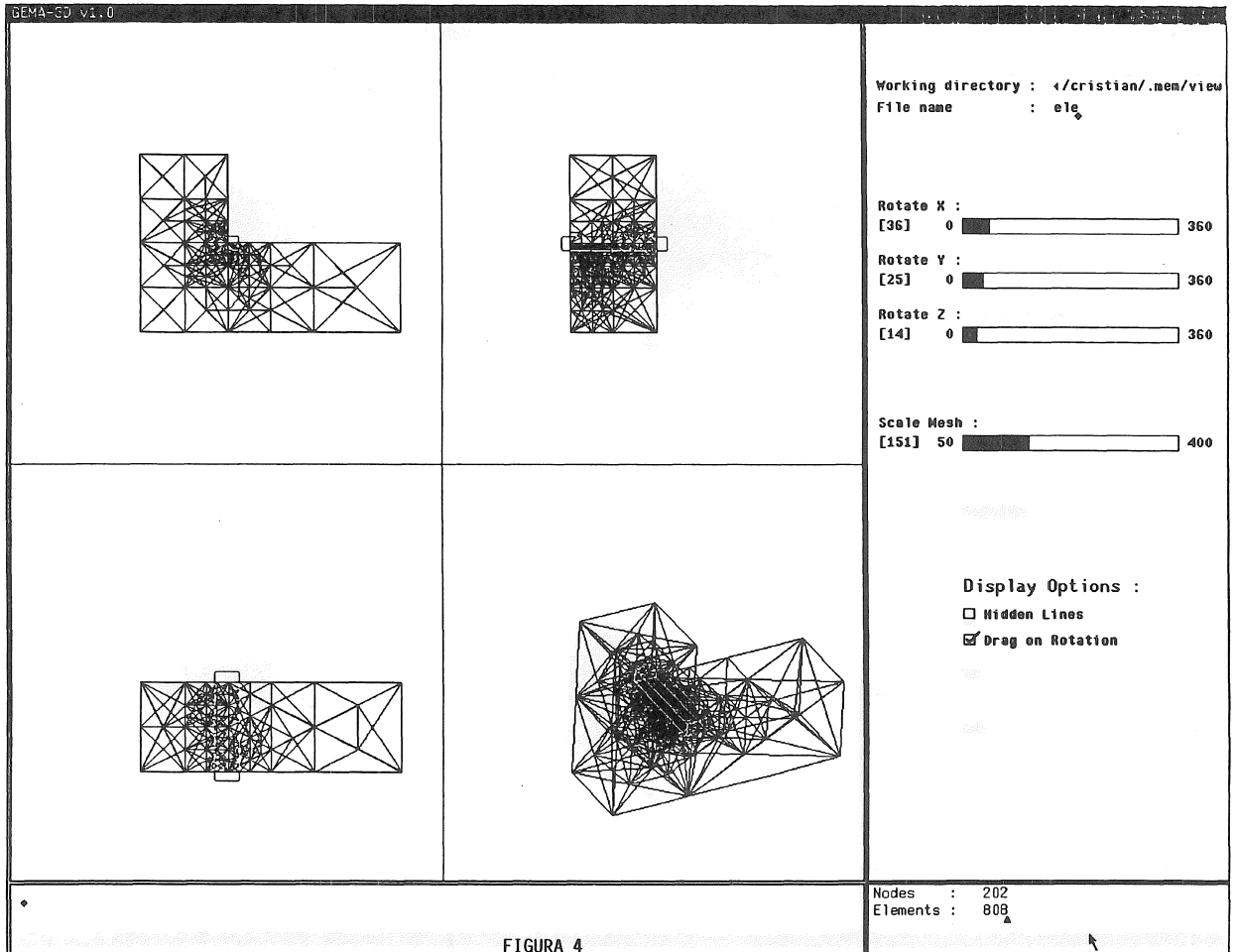

GEMA-GD v1.0
Working directory : ‹/cristian/.mem/view
File name : ele
Rotate X :
[36] 0 360
Rotate Y :
[25] 0 360
Rotate Z :
[14] 0 360
Scale Mesh :
[151] 50 400
Display Options :
Hidden Lines
Drag on Rotation
Nodes : 202
Elements : 808

FIGURA 4

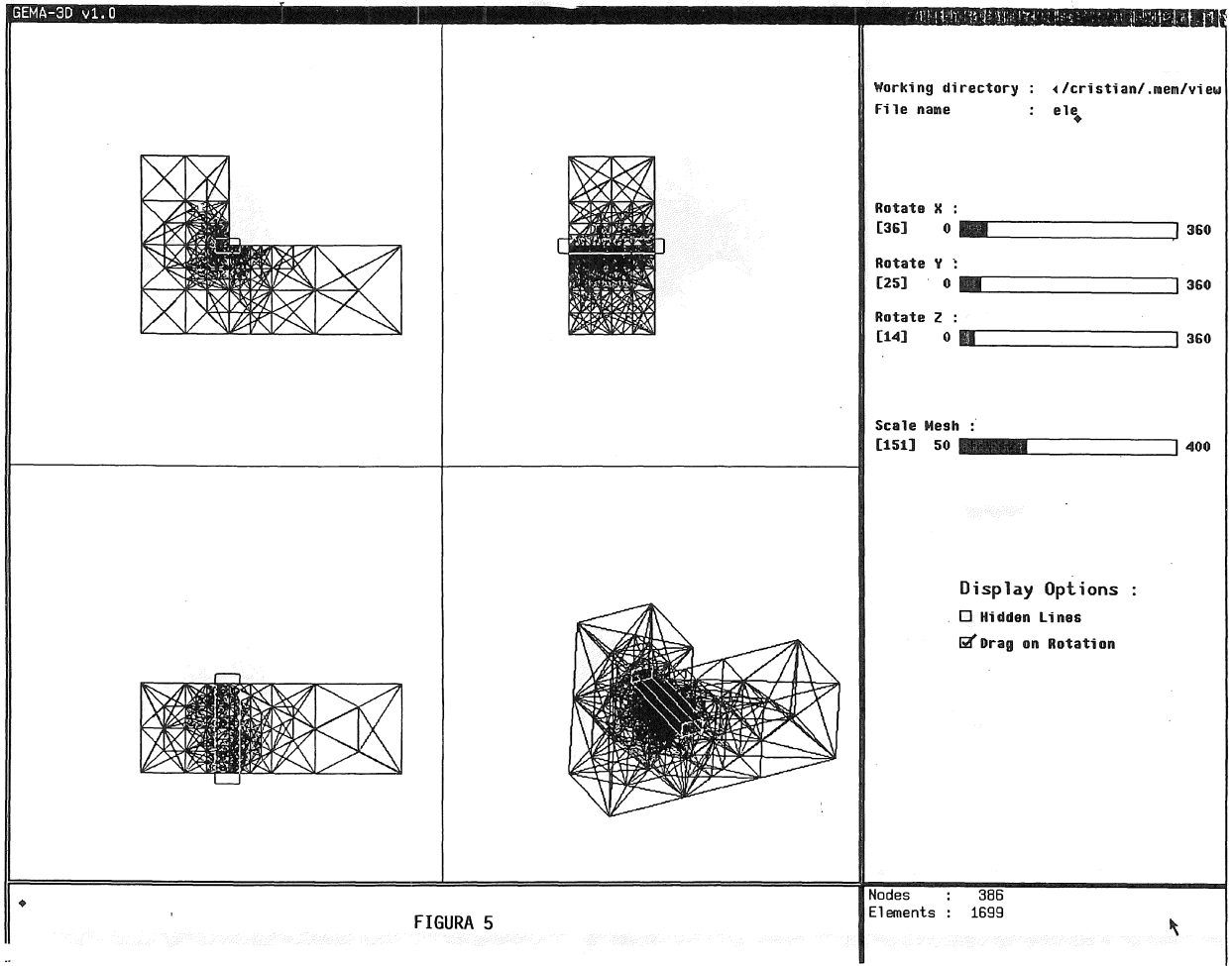

GEMA-3D v1.0
Working directory : ‹/cristian/.mem/view
File name : ele
Rotate X :
[36] 0 360
Rotate Y :
[25] 0 360
Rotate Z :
[14] 0 360
Scale Mesh :
[151] 50 400
Display Options :
Hidden Lines
Drag on Rotation
Nodes : 386
Elements : 1699

FIGURA 5

EMPAQUETAMIENTO DE PATRONES. ALGORITMO DE POSICIONAMIENTO EN FRONTERA.

Ramón Rizo Aldeguer

Departamento de Sistemas Informáticos y Computación
Universidad de Alicante.
Apdo. 99. Alicante. España.
Telf. 34-96-5903681
EARN: SIC@EALIUN11.EARN

Alfons Crespo Lorente

Departmento de Ingeniería de Sistemas, Computadores y Automática.
Universidad Politécnica de Valencia.
Apdo: 22012 Valencia. España.
Telf.: 34-96-3877571
Email: acrespo@aii.upv.es

INDICE:

Introducción. Trabajos previos. Formulación. Condiciones de no-intersección. El Modelo Radial. Algoritmo de Posicionamiento en Frontera. Resultados. Conclusiones. Referencias.

Abstract

Un problema clásico de diseño asistido por computador es la manipulación de formas bidimensionales optimizando su empaquetamiento. A lo largo del trabajo se aborda el estado del arte, particularizando en los aspectos relacionados con la línea de investigación, se revisan heurísticas empleadas en otros trabajos: cierres rectangulares y deslizamientos. Después de la formulación del problema; en busca de soluciones heurísticas, se hace una revisión de los modelos de representación, aportando el modelo radial, que sirve de soporte para razonamientos geométricos basados en la observación del comportamiento ante el problema por parte de los especialistas humanos. Para finalmente proponer el algoritmo de posicionamiento en frontera, y el análisis comparativo de resultados, donde se evalúan los resultados obtenidos. Después de las conclusiones, se proponen las líneas de continuación.

Palabras Clave

CIM, empaquetamiento de patrones, razonamiento geométrico, posicionamiento en frontera.

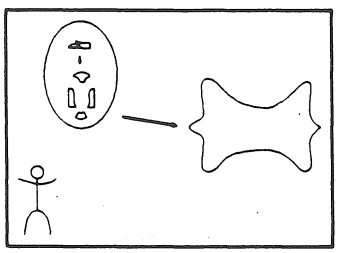
EMPAQUETAMIENTO DE PATRONES. ALGORITMO DE POSICIONAMIENTO EN FRONTERA.

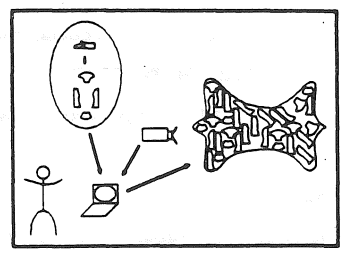
1.- Introducción.

La motivación practica del trabajo es el problema del corte de pieles en el sector del calzado, en el que se pretende optimizar el aprovechamiento de la superficie del material.

Aunque recientemente han aparecido sistemas para el posicionamiento, los de más exito son los interactivos, donde el usuario incorpora la heurística de colocación, realizándose el posicionamiento de forma manual, ayudado por un sistema CAD.

El trabajo aportado podría integrarse dentro de un sistema industrial completo, que implica areas, como son: visión, empaquetamiento de patrones, control numérico.





Este trabajo se dirige, en particular, al posicionamiento empaquetado de perfiles curvilíneos sobre zonas irregulares, de manera que la pérdida sea mínima.

La irregularidad de los perfiles en la industria del calzado es el resultado de utilizar estructuras planas para cubrir superficies tridimensionales, aquí la pérdida no se puede evitar cuando se colocan los perfiles.

El problema del empaquetamiento (semicobertura) bidimensional puede plantearse como sigue: dados un material bidimensional, un conjunto de formas bidimensionales para situar sobre el material y un conjunto de restricciones encontrar una solución que haga el mejor uso posible del material y que satisfaga todas las restricciones.

En este trabajo se aporta una solución al problema del empaquetamiento, formulado a partir de las semicoberturas homogéneas, basando dicha solución en razonamientos geométricos vinculados al comportamiento de expertos humanos, abordándose aspectos abiertos por trabajos anteriores.

2.- Trabajos previos.

El problema básico de la colocación de formas sobre otras de mayor tamaño ha sido abordado en diversos trabajos, uno de los primeros intentos de resolver mediante computador el empaquetado, se realizó por Haims en 1966 [HAIM66] y Art en 1967 [ART_67]. Proponían soluciones heurísticas basadas en cierres rectangulares y envolturas.

Este problema, por su naturaleza, no ha sido completamente resuelto, aportándose soluciones parciales que corresponden a diversas demandas industriales (Textil, metal, electrónica).

Uno de los trabajos mas recientes corresponde a Amaral, Bernardo y Jorge [AMAR90], se hizo en el marco del textil y utiliza la colocación por deslizamiento.

Los enfoques prácticos que se han aplicado pueden clasificarse en tres grupos:

- Resolución en programación lineal [KONO81]. Supuestos de rectangularidad en la zona de posicionamiento y en términos de minimización de longitud.
- Resolución mediante cierres rectangulares.
- Resolución mediante deslizamiento de las piezas a colocar.

Para la colocación basada en cierres rectangulares Adamowicz y Albano, proponen una solución en dos etapas, convirtiendo el problema original de colocar piezas irregulares en otro consistente en colocar módulos rectangulares. Para la simplificación utilizan un algoritmo de cierre para piezas irregulares en rectángulos de area mínima, el algoritmo de cierre acepta un conjunto de perfiles y produce una lista de módulos rectangulares de area mínima, su instrumentación permite la interacción con el usuario

Para la colocación basada en deslizamiento, realizada por Amaral y otros [AMAR90], la zona donde deben realizarse los cortes es un rectángulo de anchura fija y longitud infinita, la optimización viene determinada por la menor longitud usada posible. Para colocar cada pieza, establecen un vector de deslizamiento, procediendo a desplazar la pieza en su dirección. Una vez que se produce la primera colisión, se evalúan las condiciones de finalización, si alguna de ellas se cumple, el proceso finaliza y el perfil queda situado en la posible colisión. En otro caso se calcula un nuevo vector de deslizamiento, cuya dirección depende de la posición relativa de los vértices y los lados presentes en la colisión, repitiéndose el proceso mientras no se produzca alguna condición de finalización.

De las investigaciones previas del problema que se considera, aparecen los siguientes aspectos no abordados:

- Giro de las piezas. Representa básicamente un fuerte aumento de la complejidad computacional.
- Zonas de posicionamiento genéricas (cóncavo-convexas). Esta consideración afecta a todas las soluciones previas, puesto que la rectangularidad de la zona de posicionamiento sirve como base para determinar los criterios de optimización que se proponen.
- Criterios de optimización basados en consideraciones morfológicas, ocupando zonas mas alejadas del centroide, areas mínimas, etc.
- Criterios de optimización basados en el comportamiento del experto humano, en definitiva captar y reflejar la estrategia del especialista en posicionamiento manual.
- Abordar el problema con técnicas de procesamiento paralelo.
- Clasificación del problema.

3.- Formulación.

Una primera aportación de este trabajo es la formulación del problema de las semicoberturas homogéneas óptimas, Para ello, veamos unos conceptos previos.

Diremos que una colección de conjuntos forma un empaquetamiento (semicobertura) dentro de otro conjunto S, si los conjuntos son disjuntos dos a dos y su unión esta contenida, dentro de S.

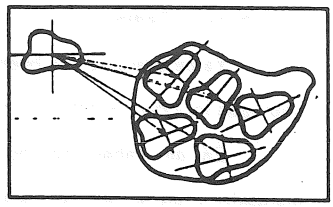
Llamaremos región bidimensional o pieza plana caracterizada por Q, correspondiente a una zona discreta del plano, a un conjunto de puntos que satisfacen la propiedad Q.

$$PP = \{ (x,y)\ R^2 / \ (x,y) \text{ satisface } Q \}$$

Llamaremos instancia de pieza plana, a cualquier transformación (traslación, giro o composición) de la pieza plana.

$$f(PP) \ / \quad f \in T_{R2}$$

Dadas dos piezas planas, diremos que una este contenida en otra, si existe una transformación del plano cuya imagen se incluye en la segunda.



PP0 y PP1 piezas planas PP0 C PP1 if $f \in T_{R2}$ / f(PP0) $\subset$ PP1.

Llamaremos Semicobertura homogénea de una región bidimensional a partir de la pieza plana, a cualquier conjunto de instancias de la pieza plana, contenidas en la región bidimensional y disjuntas dos a dos.

Diremos que una semicobertura homogénea es óptima si su cardinal es mayor o igual que el de cualquier otra que se pueda formar.

PP0 y PP1 piezas planas:

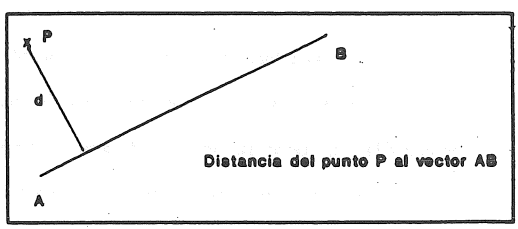
$SCH_{PP1,PP0} = \{ f_i(PP0) / f_i \in T_{R2}, f_i(PP0) \subset PP1, \text{disjuntas}\}$.

Diremos que una semicobertura homogénea es óptima (SCHO) si su cardinal es mayor o igual que el de cualquier otra que se pueda formar.

$SCH_{PP1,PP0}$ es óptima si $card(SCH_{PP1,PP0}) > card(SCH'_{PP1,PP0}) \quad \forall SCH'_{PP1,PP0}$.

4.- Condiciones de intersección y no solapamiento.

El establecimiento de las condiciones de no intersección entre polígonos se hace en dos etapas. La primera etapa define las condiciones necesarias y suficientes para intersección de lados usando las funciones D.



La D función D establece la relación posicional de un punto P respecto a un vector AB,

$A(X_A,Y_A)$, $B(X_B,Y_B)$ and $P(X_P,Y_P)$

$$D_{ABP} = \text{sign}\,[(X_A - X_B)(Y_A - Y_P) - (Y_A - Y_B)(X_A - X_P)]$$

Dependiendo de su signo el punto esta a la izquierda ($D_{ABP} > 0$), sobre la dirección ($D_{ABP} = 0$) o a la derecha ($D_{ABP} < 0$) .

A partir de la función D se pueden establecer las condiciones para intersección de dos vectores. Y también se pueden establecer las condiciones de no intersección de dos polígonos.

5.- El Modelo Radial.

Ya en busca de una buena solución heurística del problema de las semicoberturas, se han planteado los criterios para la elección del modelo de representación de los objetos del problema.

En un trabajo anterior [RIZO91], se detallan diversos modelos de representación analizados e inconvenientes para su aplicación en la propuesta realizada.

Para representar los elementos geométricos que intervienen en el problema de las semicoberturas, se parte del código de contorno de los objetos (la obtención del código de contorno puede hacerse mediante un recorrido dirigido, después de realizar un filtrado de imagen).

En el problema aparecen tres elementos a modelizar:

* Uno correspondiente a la pieza que se ubica repetidas veces (mediante instancias de clase) en la zona de recorte.
* Otro corresponde a la zona de recorte.
* Y el tercero a la colección de piezas ya ubicadas y su complemento (zona disponible para la ubicación de otras). Constituyendo el núcleo de las soluciones que se aportan dentro un objeto dinámico, cambiante en su forma, que debe proporcionar operaciones sobre intersección, inclusión y extensión.

El modelo de representación que se aporta, el modelo radial, ofrece las prestaciones necesarias para satisfacer las demandas del problema SCHO.

El modelo radial proporciona una forma de representación poligonal concavo-convexa aproximada, que tiene sus vértices de aproximación sobre direcciones predefinidas.

Suponemos una restricción inicial para todos los objetos a representar, que se denomina "propiedad de funcionalidad radial". Exigiendo el cumplimiento de esta propiedad, el objeto a representar tendrá un solo punto del contorno asociado a cada dirección que se considere tomada desde el centro de gravedad del objeto.

Se denomina centro de la representación, al centro de gravedad del objeto.

Se denomina orden de la representación, al numero de direcciones radiales que considere (2^{m+3} $m>0$).

Semirrecta de aproximación, es aquella que parte del centro y corresponde con alguna dirección radial. Sobre las semirrectas se establecen los vértices y los intervalos radiales de la representación.

Se pretende obtener un polígono de aproximación al objeto, exterior si es pieza a colocar e interior si es zona de posicionamiento.

La representación radial externa de la pieza con F radios, se define de la siguiente forma:

Una terna: vector de radios, vector de vértices y centro, tal que radios y vértices se obtienen a partir del contorno de las piezas, por deslizamiento de los vértices radiales, sobre cada par consecutivo de semirrectas de aproximación, hasta que en cada caso el segmento correspondiente a los vértices no corte al contorno.

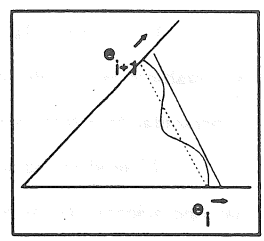
P pieza ---> $V_p = (p'_1, p'_2, .., p'_m)$

$RRB_P (F) = (r, e, CG_p)$

$r = (r_1, r_2, .., r_F)$ Radios

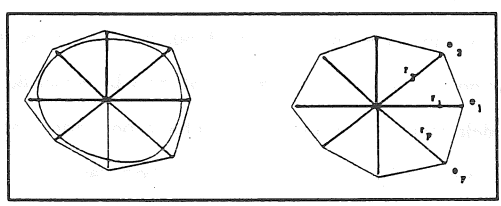
$e = (e_1, e_2, .., e_F)$ Vertices

$\forall\ p'_i \in V_P \qquad D_{e_i, e_{i+1}, p'_1} > 0$



Para ello exigimos que todos los puntos del contorno queden a la izquierda del vector e_i, e_{i+1} y si no, se desplazan e_i, e_{i+1} hacia el exterior.

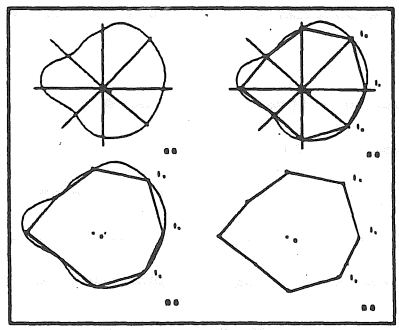
El error en la representación es la diferencia entre el area de la representación radial exterior y la del objeto, si no fuera aceptable, se considera un orden superior, hasta que lo sea.



La representación radial interna, corresponde a la zona de posicionamiento.

Se define de la misma manera que la exterior, salvo la determinación de los vértices y radios, que se hace por deslizamiento hacia el interior y exigiendo que los puntos del contorno

queden situados a la derecha del vector correspondiente a cada lado de la representación.



La no exigencia de la propiedad de funcionalidad radial, lleva a representaciones radiales mas complejas donde el vector de radios se transforma en un vector de listas de intervalos radiales y el vector de vértices en un vector de listas de vértices. Correspondientes en cada caso a cada semirrecta de aproximación.

La posición relativa de los intervalos de contorno, respecto de los lados de aproximación, se establece por medio de la función D, exigiendo que se sitúe a la izquierda en el caso de representación interna y a la derecha para externas.

El modelo radial, se ha instrumentado con la clase Repre-radial, ofreciendo entre otras las operaciones básicas para el posicionamiento, inclusión, traslación y giro, intersección, etc.

6.- El algoritmo de Posicionamiento en Frontera.

Prácticamente la totalidad de las soluciones propuestas en trabajos previos, tienen como zona de posicionamiento un rectángulo: el tejido donde hacer recortes, la placa de metal a cortar, etc. La irregularidad de la zona de posicionamiento afecta a la técnica de empaquetamiento. Como se verá aquí, el modelo radial, la representación radial proporciona una buena herramienta para la obtención de soluciones del problema de las semicoberturas.

Este apartado se organiza en dos partes: en la primera, se plantean las estrategias para el posicionamiento deducidas de la observación del comportamiento de expertos humanos durante la resolución; en la segunda parte, se presenta el algoritmo para el posicionamiento basado en el modelo radial y que refleja la estrategia de posicionamiento en frontera.

Después de observar la "*forma de hacer*" de expertos humanos situados ante al problema del recorte de pieles en el calzado, se ha intentado captar las estrategias de razonamiento geométrico que conducen su actividad. De esta observación, se deducen tres tipos de posicionamiento:

(1) **Posicionamiento compactando**, básicamente componen aglomeraciones de piezas, con la intención de que el aglomerado resultante de una nueva incorporación, tenga la menor superficie posible.

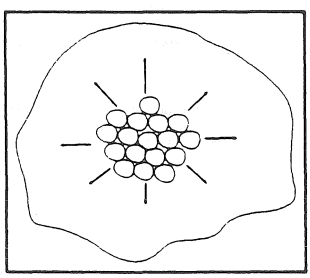
(2) **Posicionamiento adherente al contorno exterior**, en definitiva pretenden que la pieza a colocar se adapte lo más posible al contorno de la zona de posicionamiento y produzca la menor pérdida posible.

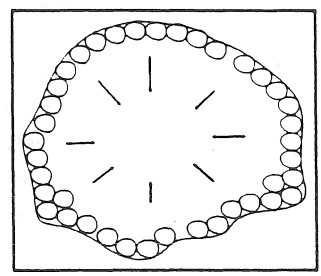
(3) Posicionamiento que se podría denominar "**dirigido a la composición final**", hace uso de los dos anteriores y añade un aspecto muy difícilmente capturable, reflejado en colocaciones concretas que no se ajustan a las estrategias anteriores pero que en definitiva les permite obtener buenas soluciones.

A la vista del comportamiento humano, se extraen dos estrategias antagónicas:

(1) Compactación de las piezas, produciendo acumulaciones de dentro hacia fuera, según se van posicionado lo objetos.

(2) Posicionamiento en frontera, produciendo acumulaciones de fuera hacia dentro.





La **compactación de piezas**, puede basarse a su vez en dos criterios:

(1-a) Distancia mínima entre centros de gravedad. El centro de gravedad de la pieza a colocar debe estar **lo más próximo** posible al centro de gravedad de la zona de posicionamiento.

(1-b) Envolvente de área mínima. Con lo que se pretende **ajustar** una pieza a las ya colocadas.

El **posicionamiento en frontera**, puede basarse a su vez en dos criterios:

(2-a) Distancia máxima entre el c.g. del objeto que se posiciona y el c.g. de la zona de posicionamiento.

(2-b) Mayor adherencia del objeto que se posiciona a la frontera de la zona de posicionamiento.

Estos criterios, cuyo origen está en el comportamiento humano frente al problema del empaquetamiento, se reflejan en los algoritmos que se proponen para obtener "buenas" soluciones.

Dada una zona de posicionamiento y una pieza, para obtener la semicobertura de la zona a partir de la pieza, con la estrategia de posicionamiento en frontera, utilizando el modelo de representación radial, se puede definir un espacio de soluciones con estructura de árbol, que se denomina Árbol_Soluciones, cuya creación y recorrido se realiza por el algoritmo básico Esquemáticamente **el algoritmo de posicionamiento en frontera** es:

1º) Se obtienen la representación radial interior de la zona, y la exterior de la pieza.

2º) - Crear Árbol_Soluciones.

- DRR_Z (F) <-- RR_Z (F).

3º) COLOCACIÓN DE UNA PIEZA MAS.

- Situar nuevo nodo en Árbol_Soluciones.

- Para todo i ∈ [1, F] ∩ Z^+ (hace referencia a los vértices de DRR_Z (F), hacer:

- Para todo j ∈ [1, F] ∩ Z^+ (hace referencia a los vértices de $IRRB_P$ (F), hacer:

3º-a) - fijar i,j

3º-b) - Crear una instancia de RRB_P (F) que se denominará $IRRB_P$ con orientación i haciendo coincidir el vértice e_j con el e_i de DRR_Z. ATENCIÓN: desplazar $IRRB_P$ hacia el interior de DRRZ, convenientemente para que $IRRB_P$ C DRR_Z.

3º-c) Modificar la representación radial de DRRZ de tal manera que excluya al polígono $IRRB_P$ obteniendo $ADRR_Z$, con lo que $ADRR_Z$ varía en alguno de sus radios y extremos respecto a DRR_Z.

3º-d) Verificar la positividad de radios de $ADRR_Z$.

si cierto:

Incluir los parámetros del posicionamiento en Árbol_Soluciones.

DRR_Z <-- $ADRR_Z$

COLOCACIÓN DE UNA PIEZA MAS (acción 3º)

si falso:

¿Falla la verificación de positividad para todos los i,j de DRR_Z?

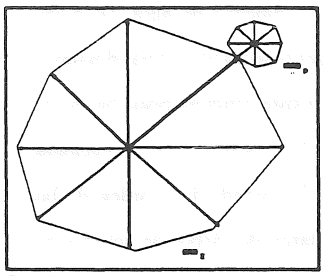
si cierto:

Detectado nodo terminal, final de rama de Arbol_Soluciones.

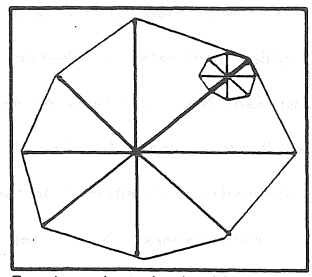
4º) Explorar Árbol_Soluciones para obtener la rama de mayor profundidad, (mayor número de piezas colocadas).

5º) Finalizar.

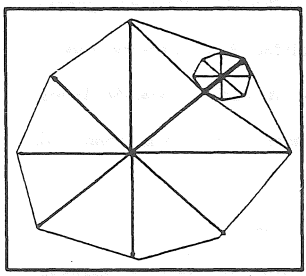
En las siguientes figuras se muestran diversas fase del algoritmo básico PF:

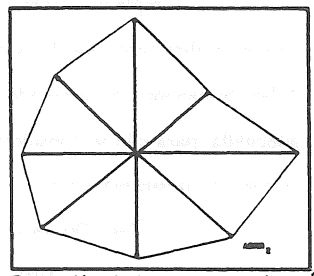


Posicionamiento exterior

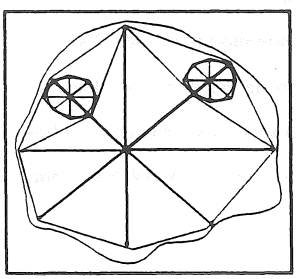


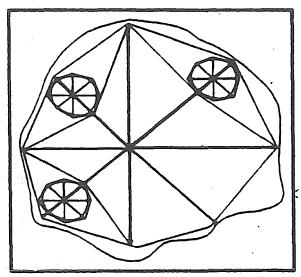
Desplazamiento hacia el interior

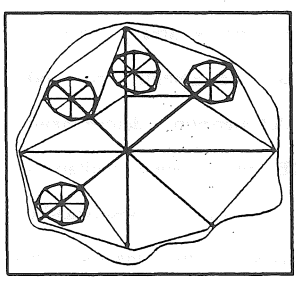


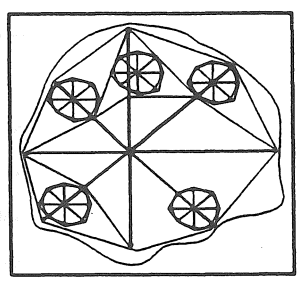


Reducción de la representación de la zona









El paso tercero del algoritmo es recursivo, provocando la construcción del árbol de soluciones, donde cada i,j definen un nodo, pudiendo tener hasta F^2 número de hijos, las hojas (nodos terminales), se obtienen cuando para ningún vértice de su representación radial dinámica asociada RR_Z (a la rama de Arbol_Soluciones hasta ese nodo) no existe ninguna posición (giro) de la pieza cuya reducción sobre RR_Z deje sus radios positivos. El espacio de soluciones exploradas es el número de ramas del árbol solución, es decir: $(F^2)^N$, siendo F el orden de las representaciones y N el número niveles del árbol. La rama más larga del árbol de soluciones, corresponde a la mejor solución de todas las exploradas.

La complejidad temporal y espacial que la creación del árbol de soluciones genera, impone utilizar criterios de poda en el algoritmo básico, con la finalidad de no explorar todo el Árbol_Soluciones. Se pretende elegir entre las K^2 posiciones posibles de cada colocación, la más adecuada, para ello se consideran dos alternativas conducentes a adherir la pieza a la frontera de la zona de posicionamiento:

2-a) *Distancia máxima al Centro de Gravedad de la zona*:

Elegir entre las K^2 posiciones, aquella que sitúa su centro de gravedad lo más lejano posible al de la zona, y verifica la positividad de radios. Entonces, crear únicamente el nodo correspondiente.

Se modifica la acción 3 del algoritmo, determinando para cada RR_P cuál es la posición de $IRRB_P$ que sitúa su centro de gravedad más lejos del centro de gravedad de RR_Z y que verifica la positividad de radios, seleccionándose esta. Recorriéndose una sola rama de Árbol_Soluciones (Lista_Solución).

2-b) *Área resultante máxima*:

Elegir entre las K^2 posiciones aquella que se sitúa de tal manera que la reduccion de representaciones radiales tiene menor área y verifica la prueba de inclusión. Entonces crear únicamente el nodo correspondiente.Se modifica la acción 3 del algoritmo, determinando para cada RR_P cuál es la posición de $IRRB_P$ que da como resultante el polígono $ADRR_Z$ de mayor área y que verifica la prueba positividad de radios, seleccionándose esta. De esta forma se obtiene una sola rama de Árbol_Soluciones (Lista_Solución).

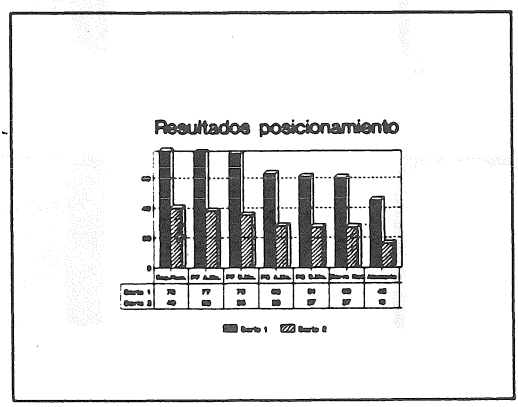
El algoritmo de posicionamiento en frontera ha sido instrumentado en Smalltalk [RIZO91].

7.- Resultados.

La medida de optimalidad de las estrategias y criterios, se ha estimado por comparación de resultados ante las mismas parejas (zona de posicionamiento, pieza)
obtenidos por las dos estrategias básicas según los criterios de poda definidos para ellas, obtenidas por experto humano, las obtenidas por un sistema de posicionamiento basado en cierres rectangulares y un sistema de posicionamiento aleatorio.

Las condiciones en que se han realizado las pruebas son:

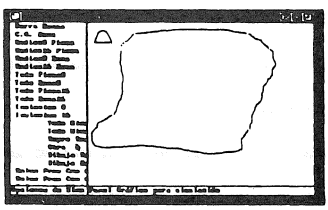
- Las parejas cumplían la propiedad de funcionalidad radial, teniendo la pieza entre 0'5 y un 5% de la superficie de posicionamiento.
-Los resultados se han tomado para las mismas parejas en cada método.
- Para la extracción de resultados del experto humano, se facilitó un sistema de posicionamiento asistido, que permitía manejar los mismos objetos que en los restantes métodos.
- Para el posicionamiento aleatorio, el posicionamiento se obtenía por generación aleatoria de un punto del interior de la zona y comprobación de no-solapamiento con los posicionados.
- Las pruebas con el sistema basado en cierres rectangulares se realizaron con un sistema cedido por una empresa vinculada al INESCOP.

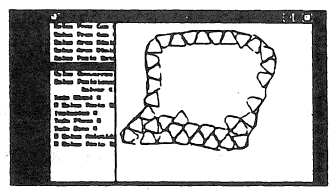


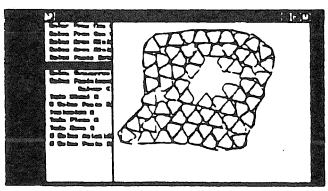
El parámetro de comparación ha sido la medida de optimización de la superficie de posicionamiento, determinada por el numero de piezas colocadas. En las tablas se muestran los resultados de dos series representativas. El posicionamiento en frontera se muestra muy próximo al de los expertos humanos, y el criterio del area máxima mejor que el de distancia máxima. Los criterios de compactación dan peores resultados, por su peor aprovechamiento de la zona exterior de posicionamiento. El Posicionamiento basado en cierres rectangulares no ha dado buenos resultados, básicamente por que no realiza giros de las piezas. El posicionamiento aleatorio se ha situado en el 50%.

En la siguiente figura se muestra los resultados relativos al posicionamiento del experto humano. Donde se ratifica el buen comportamiento del posicionamiento en frontera.

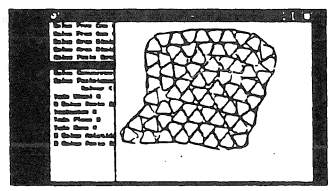
En las imágenes se muestran resultados obtenidos con el entorno de posicionamiento prototipado.



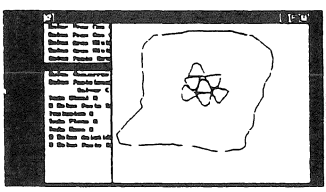




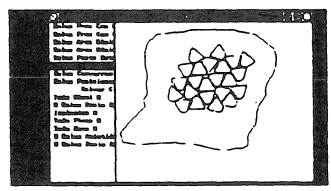
Durante el Posicionamiento en Frontera

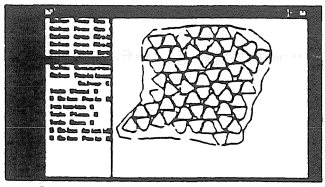


Fin Posicionamiento en Frontera.

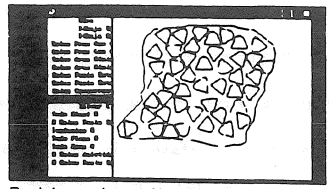


Durante copmpactación CG.





Fin Posicionamiento Compactando CG



Posicionamiento Aleatorio

8.- Conclusiones.

A lo largo de este trabajo se han revisado las soluciones propuestas por otros autores, analizándose las dos técnicas básicas que estos utilizan: El empaquetamiento mediante cierres rectangulares y el empaquetamiento por deslizamiento.

La restricción de rectangularidad condiciona su aplicación en el posicionamiento en zonas irregulares y en general anteriormente no se han considerado los giros de las piezas, debido posiblemente a que el area de aplicación en cada caso no lo exigía.

Después de formular el problema y hacer una propuesta de clasificación, se han definido las condiciones de no intersección entre polígonos utilizando la función D.

Con la intención de obtener soluciones heurísticas del problema, se han revisado modelos de representación de elementos planos. Entre otras, la consideración de irregularidad y preguntas geométricas del tipo "punto mas alejado del centro de gravedad", supusieron aportar el modelo radial, que ha servido de soporte para realizar algoritmos de posicionamiento. Para definir las estrategias de posicionamiento, se estudió el comportamiento de expertos humanos ante el problema, derivándose dos estrategias alternativas: compactación de piezas y posicionamiento en frontera. Se han aportado los algoritmos basados en cada estrategia y se han definido dos criterios específicos en cada caso, (area mínima y distancia mínima en el caso de compactación y area resultante máxima y distancia máxima en el caso de posicionamiento en frontera).

En cuanto a resultados, se ha concluido el mejor comportamiento para los posicionamientos en frontera y en particular para el criterio de area máxima resultante.

Para futuros trabajos, se plantea orientar los esfuerzos en :

- Profundizar en el modelo radial, incorporando por ejemplo, la posibilidad de definir cúmulos de piezas para realizar posicionamientos basados en estos.
- Dar mayor énfasis a la morfología de los elementos.
  - Clasificando las zonas por calidades(tratamiento de imagen).
  - Heterogeneidad y huecos. De tal manera que se reutilicen los huecos creados, para ello, las representaciones basadas en el barrido del plano pueden ser buenas candidatas.
- Incorporación de Nuevos mecanismos de razonamiento y comportamiento geométrico: vibración de piezas.
- Uso de otros modelos de computación, permitiendo manejar representaciones mas ajustadas del perfil. En esta línea podría usarse el modelo conexionista.

Referencias

[ADAM69] *Adamowicz, M.; "The Optimum Two-Dimensional Allocation of Irregular Multiply-Connected Shapes with Linear, Logical, and Geometric Constraints". Ph. D. dissertation, Elect. Engg. Dept., New York University, New York, 1969.*

[ADAM76] *Adamowicz & Albano; "Nesting Two-Dimensional Shapes in Rectangular Modules". Computer Aided Design, Vol 8 No. 1, January 1976.*

[ALBA80] *Albano, A. & Sapuppo, G.; "Optimal Allocation of Two-Dimensional Shapes Using Heuristic Search Methods". IEEE Trans. on Systems, Man, and Cybernetics, Vol. 10, No. 5, May 1980, pp. 242-248.*

[AMAR90] *Amaral; Bernardo & Jorge; "Marker-Making Using Automatic Placement of Irregular Shapes for the Garment Industry". Computers and Graphics, Vol 14, No. 1, 1990.*

[ART_67] *Art; "Computerized Marker Making: An Approach to the Pattern Layout Problem in the Garment Industry". The Journal of the Apparel Research Foundation, Vol 2, No. 2, 1967*

[FORT85] *Fortune, S.J.; "A Fast Algorithm for Polygon Containment by Translation" ICALP, 85.*

[HAIM66] *Haims, M.; "On the Optimum Two-Dimensional Allocation Problem". Ph. D. dissertation, Elect. Engg. Dept., New York University, New York, 1966.*

[KONO81] *Konopasek, M.; "Mathematical Treatments of Some Apparel Marking and Cutting Problems". U.S. Dept. of Commerce Report 99-26-90857-10, June 1981.*

[LING83] *Lingas, Andrzej; "Heuristics for Minimum Edge Length Rectangular Partitions of Rectilinear Figures". LNCS N.145, Ed. A. B. Cremers & H.P. Kriegel, Springer Verlag, 1983.*

[MAHA84] *Mahadevan, A.; "Optimization in Computer Aided Pattern Packing" Ph. D. dissertation, North Carolina State University at Raleigh, 1984.*

[PARB87] *Parberry, I.; "Parallel Complexity Theory". John Wiley & Sons, Inc.,1987, New York, Toronto*

[RIZO91] *Rizo, R.; "Semicoberturas Homogéneas Bidimensionales" Ph. D. disertation, Universidad Politécnica de Valencia. Spain, 1991.*

[SIRJ88] *Sirjani, Abolfazl & Cross, George R; "An Algorithm for Polygonal Approximation on a Digital Object". Pattern Recognition Letters 7, 1988, pp. 299-303.*

PERFORMANCE OF A PHONETIC ENCODING SCHEME FOR SPEECH RECOGNITION USING NEURAL NETWORKS

V. Rodellar, P. Gómez, C. García, F. Naharro, M. Pérez, y C. Gonzalo

Departamento de Arquitectura y Tecnología de Sistemas Informáticos
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo, s/n
Boadilla del Monte
28660 Madrid
Tfno: +34.1.336.73.84
Fax: +34.1.336.74.12
E-mail: pedro@pino.datsi.fi.upm.es

Abstract

Through the present work, an Encoding Scheme for the Identification of the Phonetic Features of Speech, introduced and developed in a preliminary research [1, 2], is implemented on a Time-Delay Back-Propagation Neural Network (TDBPNN), and its most relevant features are analized. The Encoding Scheme is based on an 8-bit Hamming code, and can be represented by an 8-dimensional hypercube. A separate Phonetic Subgraph, when the relevant nodes and edges are considered, is presented, showing the Minimum Distance Pairs for Spanish. The problem of using a Time Delay Neural Network to support this Encoding Scheme is addressed, pointing to the methods train the Network. For such, a Fragmentation and Labeling Technique based in the PARCORgram Correlation Matrix of Speech is presented. Results show that using a 48:9:8 Back Propagation Time-Delay Network with fragments of Speech of the kind VCV or CV from the densest subset of sounds in the Phonetic Subgraph, the percentage of failures produced during the recognition process may be substantially reduced. The technique shown may be used in Computer Aided Speech Learning.

Key words:

Speech Recognition, Neural Nets, Phonetic Coding, Spectral Estimation, Correlation

Introduction

The present paper describes an Encoding Scheme for the Phonetic Features of Spanish, which may be usefully employed in the automatic determination of the phonetic features of a given segment of Speech. The Encoding Scheme is based on an 8-bit Hamming Distance Code related to the Phonetic Features of Spanish as shown in Table 1, and may be expressed by an 8-dimensional hypercube as a fully connected graph [3], in which a given node is associated to a given sound in the Encoding Scheme.

Bit #	Feature	Comment
0	Oral/Nasal	b0 =1 oral; b0=0 nasal
1	Voiced/Unv.	b1=1 voiced; b1=0 unvoiced
2 3	Degree of closure	b2=0, b3=0; plosive b2=0, b3=1; fricative b2=1, b3=0; vowel b2=1, b3=1; semicons., glide
4 5	Articulatory place	b4=0, b5=0; palatal b4=0, b5=1; dental b4=1, b5=0; velar b4=1, b5=1; labial
6	In vowels: Rnd/Oval In consonants: Frontal/Lateral	b6=1 rounded; b6=0 oval b6=1 frontal; b6=0 lateral
7	Multiple/Simple	b7=1 multiple; b7=0 simple

Table 1. Feature Encoding Scheme

Not every node in the related graph is attached to a sound, most of them being empty. The subgraph obtained removing the empty nodes and edges may be seen in Fig. 1.

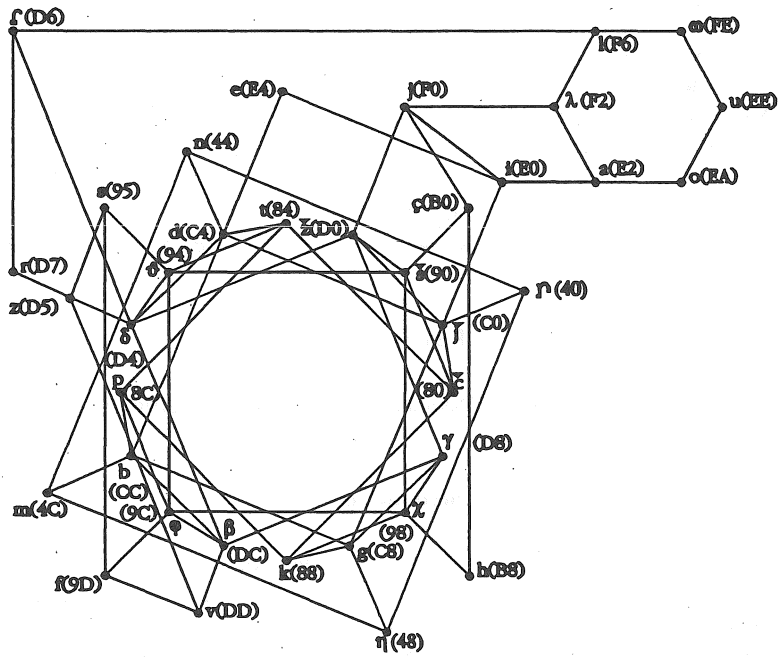


Fig. 1. Subgraph corresponding to the Encoding Scheme being proposed.

A given phoneme (shown in boldface), and its corresponding hexadecimal code, appear close to the corresponding node. The hexadecimal code is introduced as an abreviation of the related binary code, with bit #0 and bit #7 respectively taken as the most and least significant ones. In this subgraph, the edges relate sounds which may be considered familiar to others in the sense that they form a Minimum Hamming Distance Pair. This fact can be checked with sounds /t/ and /d/. Their corresponding codes are the hexadecimal numbers (H84) and (HC4). This implies a one-bit difference between them, corresponding to the bit coding the Phonetic Feature of Voicing (bit #1).

Phonetic Encoding by Neural Networks

This Encoding Scheme is specially devised to be supported by a Time-Delay Back-Propagation Neural network (TDBPNN) [4], which assigns an 8-bit code to each fragment of LPC vectorized Speech [1]. LPC extraction may be done using Blind Overlapping Fragmentation (BOF), or Phonetic Sensitive Fragmentation (PSF). Blind Fragmentation produces equal-size fragments, although is not careful with the changes in the spectral characteristics of the signal. This last fact may be attenuated using overlaping fragments. PSF is very careful with the spectral characteristics of Speech, produces a more compact set of LPC vectors, and may compare to Viterbi's Algorithms [5]. A typical BOF spectrogram obtained applying LPC extraction and spectral reconstruction may be seen in Fig. 2 for the utterance /aγa/.

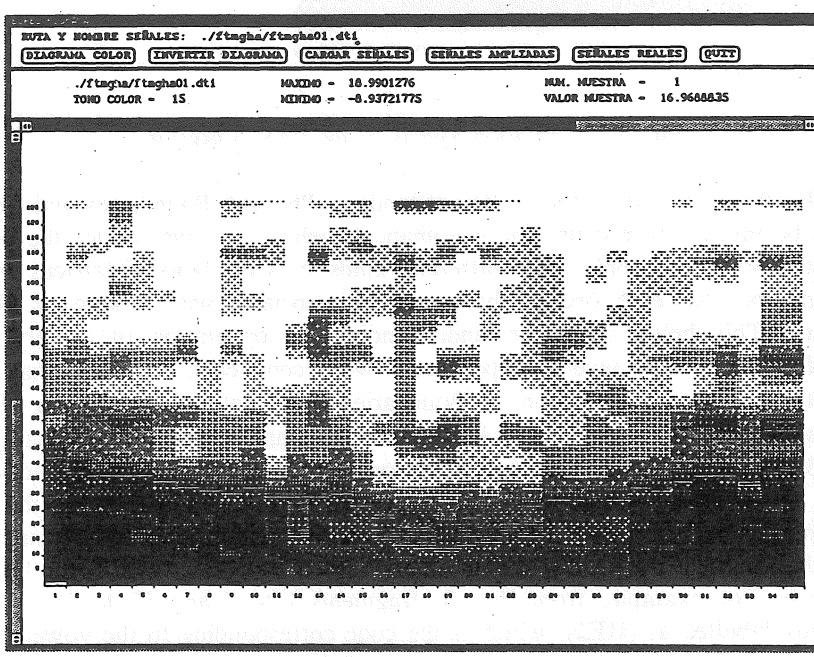


Fig. 2. Spectrogram obtained by Blind Overlaping Fragmentation of /aγa/.

Speech Labeling is a very complicated, sensitive and error-prone task, in the sense that an Expert Phonetician has to assign a tentative code to every LPC vector, which in many cases don't show clearly established Phonetic Features. To help the Expert Phonetician in this task, the present research introduces a new technique, based in using the Correlation Matrices of the PARCORgram, considered as the Time-Ordered Set of LPC vectors associated with a Speech fragment. A typical PARCORgram Correlation Matrix may be seen in Fig. 3.

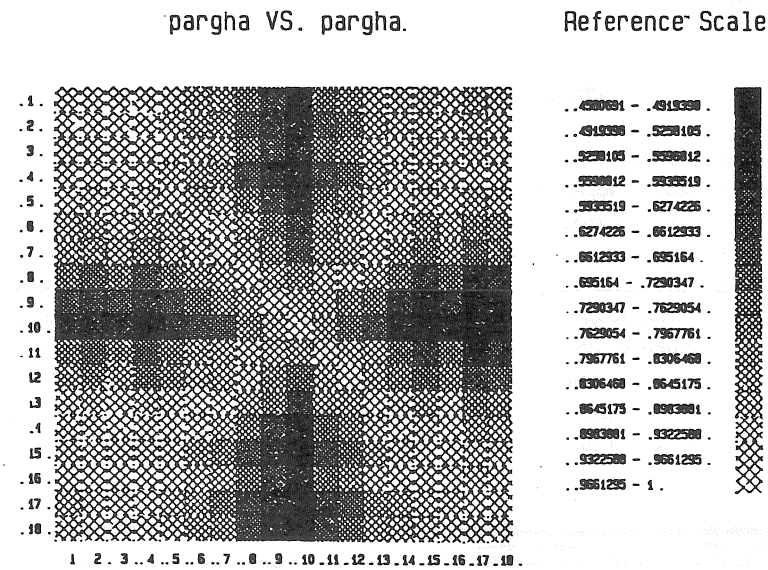


Fig. 3. Correlation Matrix of the PARCORgram.

It may be shown that only four different kinds of Phonetic Boundaries and Contours have to be taken into account for Automatic Labeling, and that under the point of view of Correlation, only four different kinds of Correlation Matrices may be expected, and that each one of them may be associated with a kind of Phonetic Boundary. This helps greatly in understanding the opperation of a very simple TDBPNN, for which several experiments were conducted, showing that in the simplest case, the identification of Boundaries and Contours by such a Network requires the use of a minimal 3:4:2 architecture, and demonstrating that Single-Layered Neural Networks, such as Perceptrons or ADALINES [6, 7] are not able of solving the problem, which can be considered as a generalization of the XOR-Problem. The study of the Correlation Matrices gives the hints to the Phonetician of where the Phonetic Boundaries should be established, to correctly label the fragments of Speech. For example, from Fig. 3, fragments 1 to 7 and 13 to 18 should be apparently labelled as (HE2), which is the code corresponding to the vowel /a/, and fragments from 8 to 12 should be labelled as (HD8), which is the code assigned for /γ/. A special Network may be devised for this purpose, relieving the Phonetician of such a cumbersome task. The second part of the paper is devoted to describe the behaviour of an Encoding TDBPNN, showing that an architecture of 48:9:8 as the

one in Fig. 4, and a Correlation-Matrix-Based Labeling, produce important improvements over preliminary approaches [2].

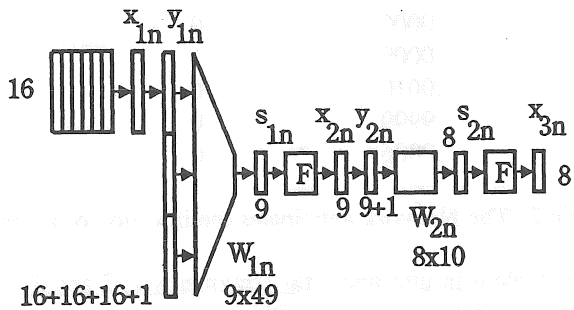


Fig. 4. Architecture of the TDBPNN being used

In Fig. 4, W are the weight matrices, x are the normalized inputs to each layer, and s are the unnormalized outputs. F is the normalizing nonlinear function. The dimensions of the matrices and vectors is stated in the lower part of the figure.

Results

The performance of such structure is checked using a core of 16 "cardinal" consonants, from the densest part of the Encoding Graph shown in Fig. 1, plus a vowel, in groups with structure VCV or CV. The use of the cardinal set of consonants, which is in itself a 4-dimensional hypercube, allows to test the performance of the Network to differentiate between Minimum Distance Pairs, in the sense of Hamming Distance, and as such allow a very strict check of the behaviour of the Network under the worst conditions. According to these considerations, a set of experiments was conducted using the cardinal set formed by the consonants /p/t/c/k/b/d/j/g/φ/ϑ/s/χ/ coarticulated with the vowel /a/, in structures of the kind CV, and the consonants /β/δ/z/γ/ coarticulated with the vowel /a/ in structures of the kind VCV. Each structure was fragmented using PSF techniques into 10 to 20 fragments, and LPC extracted. Each 16-element LPC vector was assigned an 8 bit Phonetic Code. With three consecutive of these groups a sample block to be fed to the TDNN was built. Thus, a total number of 224 of these samples were used during the training process. The 48:9:8 TDBPNN was trained using this set of samples during 935 training steps. When the process was completed, the set of samples associated to a given sound, was presented to the Network for its recognition, and the Network's response was compared to its corresponding Phonetic Label. Then, it was found that only 5 bits were erroneously coded, out of the total amount of 3584 checked, which yielded a percentage of 97,7% Phonemes correctly encoded. Besides, most of the observed mistakes were not fatal, in the sense that three of them appeared in the boundary between consonant and vowel, as exposed in Table 2. The Network "anticipated" the insertion of the vowel in disagreement with its preassigned label.

Sample 4 in utterance /ka/ (fragments 5, 6 and 7):

Bit #	Output value	Objective value	Agreement
0	.9994	1	Yes
1	.9957	0	No
2	.9999	0	No
3	.0000	0	Yes
4	.0009	1	No
5	.0010	0	Yes
6	.9999	0	No
7	.0006	0	Yes

Table 2. The Network anticipates the insertion of a vowel.

Sample 0 in utterance /ta/ (fragments 1, 2 and 3):

Bit #	Output value	Objective value	Agreement
0	.9997	1	Yes
1	.0244	0	Yes
2	.0009	0	Yes
3	.0000	0	Yes
4	.0249	0	Yes
5	.0000	1	No
6	.0008	0	Yes
7	.0002	0	Yes

Sample 1 in utterance /ta/ (fragments 2, 3 and 4):

Bit #	Output value	Objective value	Agreement
0	.9997	1	Yes
1	.0213	0	Yes
2	.0107	0	Yes
3	.0000	0	Yes
4	.0206	0	Yes
5	.0000	1	No
6	.0107	0	Yes
7	.0002	0	Yes

Table 3. Failures present in the detection of /ta/ coded as /ca/

In Table 2, an utterance of /ka/, divided into 13 samples, each one containing 3 time-consecutive fragments according to the suggestion of the PARCORgram of the Correlation Matrix, was being recognized. The Network's response for each bit in the Phonetic Code was listed under **Output value**, and the desired response (objective) was listed under **Objective value**. The response for the first four samples, from 0 to 3, not presented in the Table, showed a complete agreement between the desired response (**H88**), corresponding to /k/, and the actual response. For sample no. 4, the actual and desired responses were listed in Table 2, showing that there was a disagreement in four of the code bits, corresponding to numbers 1, 2, 4 and 6. The response produced by the Network was (**HE2**), this last code corresponding to /a/. Having into account that samples 5 to 12 were associated to that last sound, it seems

that the apparent failure was due to an anticipation in the detection of the boundary between consonant and vowel, rather than to a malfunctioning. In Table 3, the responses to the first two samples out of a set of 10, are shown. It can be seen that the actual responses yield code (H80) corresponding to the consonantal sound /c/ when (H84) corresponding to /t/, was expected. These mistakes could be produced by an excess of aspiration when uttering the original consonant, and in this case the Network's suggestion should be taken literally for the reconsideration of the assignment done during the labeling process. Finally, Table 4 shows the behaviour of the network in a typical case of error-free processing, involving the recognition of the utterance /aɣa/. It can be seen, that the detection of the vowel and the approximant, are quite complete, even in the boundaries between them, as is the case with samples 10 and 11, which share the spectral information of fragments 12 and 13.

Sample 4 in utterance /aɣa/ (fragments 5, 6 and 7):

Bit #	Output value	Objective value	Agreement
0	.9995	1	Yes
1	.9998	1	Yes
2	.9998	1	Yes
3	.0000	0	Yes
4	.0012	0	Yes
5	.0000	0	Yes
6	.9998	1	Yes
7	.0004	0	Yes

Sample 10 in utterance /aɣa/ (fragments 11, 12 and 13):

Bit #	Output value	Objective value	Agreement
0	.9977	1	Yes
1	.9898	1	Yes
2	.0009	0	Yes
3	.9987	1	Yes
4	.9997	1	Yes
5	.0077	0	Yes
6	.0009	0	Yes
7	.0019	0	Yes

Sample 11 in utterance /aɣa/ (fragments 12, 13 and 14):

Bit #	Output value	Objective value	Agreement
0	.9997	1	Yes
1	.9999	1	Yes
2	.9938	1	Yes
3	.0001	0	Yes
4	.0015	0	Yes
5	.0000	0	Yes
6	.9938	1	Yes
7	.0002	0	Yes

Table 4. Three typical fragments being properly identified.

The results rendered by the present approach show a great accuracy in the task of Phonetic Feature Encoding, and allow us to conclude that the research in progress is highly interesting because may be efficiently applied to Speech Processing and Recognition. Other approaches incorporating more sophisticated Encoding Schemes, and more powerful Network Architectures are also being considered. The applications of the present research are oriented to Computer Aided Speech Training, such as in Language Disorder Care, or in Language Schools.

Acknowledgements

The present research is being developed under Grants Nos. C072/90 from the Plan Regional de Investigación de la Comunidad de Madrid and Acción Concertada UPM A91-0020-02-78.

References

[1] V. Rodellar, P. Gómez, M. Hermida, A. Díaz and R. W. Newcomb, "A VLSI Architecture for the Support of an Auditory Model for Hearing and Speech Processing", **Proc. of the 33rd Midwest Symposium on Circuits and Systems**, Calgary, Alberta, Canada, August 12-15, 1990, pp. 787-790

[2] V. Rodellar, F. Naharro, C. García, S. Martín, M. L. Muñoz and P. Gómez, "A Neural Network for the Extraction and Characterization of the Phonetic Features of Speech", **Proc. of the Fourth Int. Conf. on Neural Networks and Applications, NEURO-NIMES'91**, Nimes, France, November 4-8, 1991, pp. 203-212

[3] A. Gibbons, **Algorithmic Graph Theory**, Cambridge University Press, Cambridge, England, 1985

[4] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano and K. J. Lang, "Phoneme Recognition using Time-Delay Neural Networks", **IEEE Trans. on ASSP**, Vol. 37, No. 3, March 1989, pp. 328-339

[5] T. Parsons, **Voice and Speech Processing**, McGraw-Hill, New York, 1987

[6] T. Khanna, **Foundations of Neural Networks**, Addison-Wesley Pub. Co., Reading, Massachusetts, 1990

[7] B. Widrow and M. A. Lehr, "30 Years of Adaptive Neural Networks: Perceptron, Madaline and Backpropagation", **Proc. of the IEEE**, Vol. 78, No. 9, Sept. 1990, pp. 1415-1442

UN ALGORITMO DE MARCHA PARA LA INTERSECCION DE DOS SUPERFICIES*

OMAIRA RODRIGUEZ
ESCUELA DE COMPUTACION
FACULTAD DE CIENCIAS
UNIVERSIDAD CENTRAL DE VENEZUELA

APARTADO 47002
LOS CHAGUARAMOS 1041 -A
CARACAS - VENEZUELA

TELEFONO: + 58-2-662-7233
FAX: + 58-2-662-7121
E-mail: orodrig@dino.conicit.ve

*Trabajo parcialmente financiado por CONICIT

ABSTRACT. Se propone un algoritmo heurístico para encontrar la curva intersección de dos superficies dadas en forma implicita. Los puntos sobre la intersección se obtienen paso a paso haciendo marchar un cubo sobre la curva.

KEY WORDS. Geometría, marcha, superficies, curva, intersección.

1. INTRODUCCION

Cualquier sistema diseñado para manejar superficies, necesita producir representaciones de curvas que yacen sobre esas superficies, tales como el contorno de datos experimentales, intersección de superficies y cálculos de curvas siluetas para la producción de dibujos. Por su interés computacional, matemático, así como su amplia aplicación este tópico ha recibido mucha atención en los últimos años.

Se propone un algoritmo heurístico para obtener la curva intersección de dos superficies conocidas, dadas en forma implícita, en una región del espacio.

Nuestro algoritmo usa una técnica simple de marcha para 3D que rastrea la curva intersección de dos superficies definidas en forma implicita

$$f(x,y,z) = 0,\ g(x,y,z) = 0$$

Se imponen ciertas condiciones de regularidad sobre la curva intersección: que la intersección sea transversal, que la curva sea simple y conexa.

La selección de los puntos se realiza paso a paso, en cada uno de ellos se determina una dirección de movimiento y se selecciona aquel punto que se encuentre mas cercano a, o sobre, la curva intersección, de esta manera se obtiene una secuencia de puntos que describen la curva, o segmento de ella.

2. DESCRIPCION DEL ALGORITMO

Sea P_0 un punto inicial, el cual puede ser conocido o derivado por otro método, perteneciente a la curva intersección de las dos superficies f y g dadas en forma implícita

$$f(x,y,z) = 0$$
$$g(x,y,z) = 0 .$$

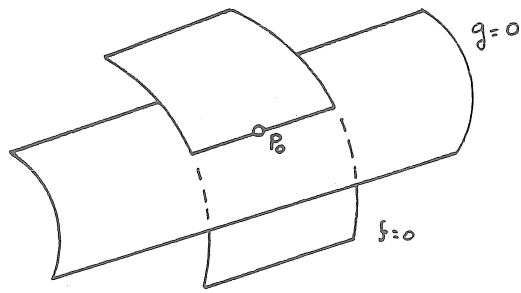


Fig. 1

Consideremos un cubo cuyo centro es el punto inicial $P_0(x_0,y_0,z_0)$ Fig. 2, es decir, la curva intersección pasa por el centro del cubo. Nos interesa entonces determinar el próximo punto de la curva, o lo que es lo mismo, encontrar el punto mas cercano a la curva intersección y además determinar la dirección de movimiento.

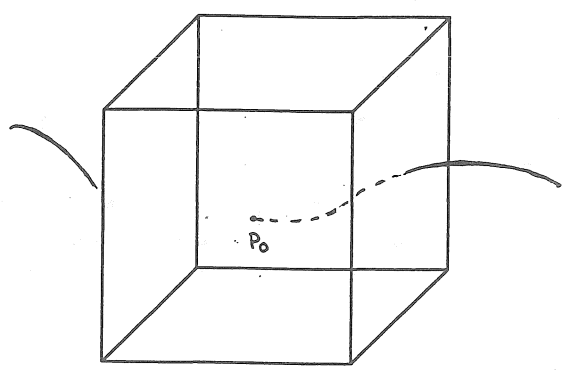


Fig. 2

Cada uno de los vértices, centros de cara y puntos medios de aristas son posibles candidatos a ser próximos puntos de nuestra curva Fig. 3.

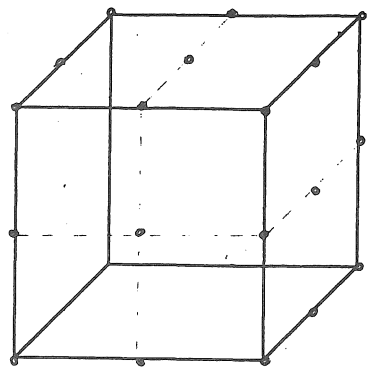


Fig. 3

Bajo nuestra suposición tenemos en total 26 candidatos a ser próximos puntos. Las ternas (i, j, k) i = -1, 0 , 1; j = -1 , 0 , 1 ; k = -1 , 0, 1 nos permiten identificar cada uno de los vértices, centros de cara y puntos medios de las aristas. Cada uno de los 26 candidatos pertenecen a una y solo una de estas ternas de puntos. A su vez sirven para identificar la dirección de movimiento a lo largo de la curva, dada por (dirX, dirY, dirZ).

Así pues (dirX, dirY, dirZ) = (1, -1, 1) nos indica que el próximo punto mas cercano a la curva será

$$(X_{old} + \text{delta } X,\ Y_{old} - \text{deltaY},\ Z_{old} + \text{deltaZ})$$

donde deltaX, deltaY, deltaZ son los respectivos incrementos a las correspondientes variables en R^3 y X_{old} , Y_{old}, Z_{old} es la posición actual del centro del cubo.

Para la selección del primer punto se evaluan las funciones f(x,y,z) y g(x,y,z) en cada uno de nuestros candidatos a próximo punto, para el primer paso 26 puntos. Nuestro próximo punto P_{next} es aquel que minimice

$$|f(x,y,z)| + |g(x,y,z)| \qquad (1)$$

Efectivamente, si (x* , y*, z*) es un cero de la curva intersección, entonces f(x*,y*,z*) = 0 y g(x*,y*,z*) = 0 por lo tanto es razonable suponer que si el punto (x,y,z) minimiza |f|+|g| éste será usualmente el más cercano a la curva buscada. La selección de este punto nos determina la dirección inicial de curva En caso de que más de un punto satisfaga la condición, arbitrariamente seleccionamos uno de ellos. Esta selección puede ser mejorada como se explicará mas adelante.

Teniendo estos dos puntos, el punto inicial dado (centro del cubo P_{old}) y el primer punto escogido P_{new} , podemos obtener un próximo punto cercano a la curva moviéndonos (marchando) al próximo cubo, como se explica a continuación.

Consideremos ahora que el punto P_{new} es el centro de nuestro cubo, luego dependiendo de su procedencia podemos establecer los siguientes casos, en cada uno de ellos hemos limitado el número de puntos a considerar como candidatos a próximos puntos.

i) P_{new} proviene de un centro de cara, del cubo cuyo centro es P_{old}.

ii) P_{new} proviene de un vértice, del cubo cuyo centro es P_{old}.

iii) P_{new} proviene de un punto medio de arista, del cubo cuyo centro es P_{old}.

Para el caso i) los candidatos a ser próximos puntos más cercanos a nuestra curva, se reducen a aquellos que se encuentran en la cara de enfrente (del próximo cubo) a la dirección de movimiento Fig. 4.

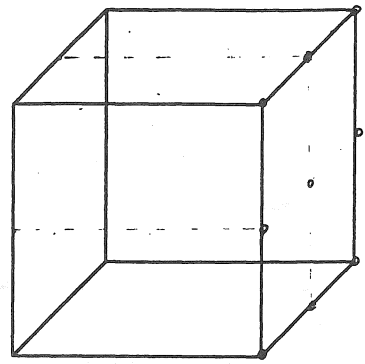


Fig. 4

Para seleccionar el nuevo punto evaluamos las funciones f y g en cada uno de los nueve puntos candidatos, el próximo punto será aquel que satisfaga la condición de minimización (1).La dirección de movimiento (dirX, dirY, dirZ) viene dada por la posición , en el cubo, del punto seleccionado.

Si el centro del cubo proviene de un vértice (caso ii) solo nos interesan aquellos puntos que estan situados en la esquina diagonal opuesta al vértice de procedencia.

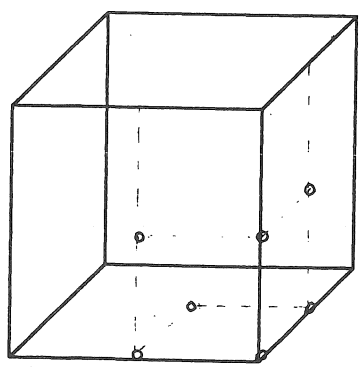


Fig. 5

P_{next} se obtiene al evaluar las funciones f y g en cada uno de estos 7 puntos, el punto seleccionado es el que satisface la condición de minimización (1). De nuevo, la dirección de movimiento

(dirX,dirY,dirZ) esta dada por la posición, en el cubo, que ocupa el punto seleccionado.

El último caso es cuando el centro del cubo proviene del punto medio de una arista (caso iii), aquí los candidatos a próximos puntos son aquellos situados en la esquina contigua,opuesta en diagonal, a la arista de procedencia. Al igual que en los otros casos la dirección de movimiento (dirX,dirY,dirZ) esta dada por la posición que en el cubo ocupa el punto seleccionado.

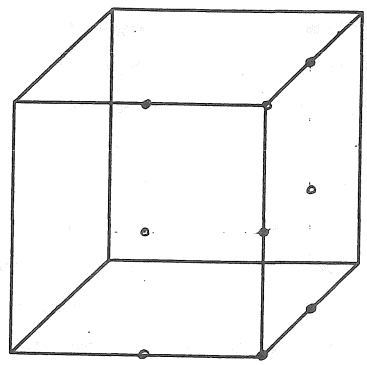


Fig. 6

En general, si tenemos dos puntos consecutivos de la curva: P_{old} y P_{new} , se escoge P_{next} , evaluando f y g en los puntos determinados por la dirección de movimiento dada por estos y de acuerdo a cada uno de los casos ya estudiados. El proceso continua actualizando las posiciones

$$P_{old} = P_{new}$$
$$P_{new} = P_{next}$$

3. ALGUNAS CONSIDERACIONES

La primera vez que se genera un próximo punto se necesitan 26*2 evaluaciones de la función, en los siguientes puntos el máximo número de evaluaciones llega a ser 9*2 en el peor de los

casos lo cual implica que se disminuye considerablemente el número de evaluaciones de la función por cada punto generado.

La dirección inicial va a permitir orientar el trazado de la curva, luego la selección del punto inicial y la dirección inicial juegan un papel importante en el éxito del algoritmo, se podría utilizar un procedimiento más sofisticado para esta selección, sin embargo se debe tomar en cuenta los costos computacionales que esto acarrearía.

Es razonable esperar que si (x* , y* , z*) pertenece a la curva intersección desearíamos que al menos dos de nuestros puntos seleccionados como candidatos esten sobre la curva, sin embargo esto no siempre es cierto, ya que la curva puede pasar entre estos dos puntos. También puede ocurrir que existan más de dos puntos donde f y g se anulan, esto nos índica un comportamiento irregular de la curva, estos casos los excluimos de esta discusión.

Una de las posibles mejoras de este algoritmo consiste en la modificación del criterio de selección cuando más de un punto satisface la condición de minimización por una que tome en cuenta a las funciones envueltas bien sea su derivada o cualquier otra propiedad inherente a la función.

Hemos pensado que es posible automatizar el paso de recorrido, para ello se podría utilizar la noción de continuidad geométrica que implicaría la evaluación de la función en tan solo tres puntos.

El algoritmo propuesto genera una secuencia de puntos en coordenadas de mundo que luego son desplegadas usando un mecanismo de visualización en R^3 . El usuario puede decidir el recorrido que hará sobre la curva. Así mismo puede modificar el paso del recorrido.

REFERENCIAS

1 Asteasu, C . 'Intersection of arbitrary surfaces'. Computer Aided Design. Vol 20 No 9 (November 1988) pp 533-538.

2 Chandler, R.E. 'A tracking algorithm for implicitly defined curves'. IEEE Computer Graphics & Applications Vol 9 (March 1988) pp 83-89.

INTEGRACIÓN Y COMPLEJIDAD EN SISTEMAS MULTIEXPERTOS

Alberto Sanfeliu
Instituto de Cibernética
Diagonal 647, 08028 Barcelona (Spain)

Curriculum

Alberto Sanfeliu obtuvo la titulación de Ingeniero y Doctor Ingeniero Industrial en la Universidad Politécnica de Cataluña (UPC), en 1978 y 1982 respectivamente. De 1975 a 1979 estuvo en el Instituto de Cibernética (UPC-CSIC) de Barcelona. Durante un periodo de dos años realizó trabajos de investigación en Purdue University (USA) en el departamento del Prof. K.S.Fu. Actualmente es Profesor titular en el area de conocimientos "Ciencias de la Computación e Inteligencia Artificial" en la UPC y realiza su labor investigadora en el Instituto de Cibernética. Es en la actualidad Presidente del IAPR Technical Committee on Syntactic Pattern Recognition y Presidente del Grupo Técnico de Reconocimiento de Formas (CEA-IFAC). Ha publicado en diversas revistas internacionales y nacionales y es coeditor en diversos libros y revistas. Su investigación actual se centra en temas relacionados con robótica, principalmente en visión por computador, desde la perspectiva de las areas de Inteligencia Artificial y Reconocimiento de Formas.

Resumen

Sistemas para diagnosticar averías en redes telefónicas o para interpretar escenas de robótica hacen uso de diversos especialistas que realizan tareas conjuntamente. Algunos de estos especialistas amenudo están diseñados como sistemas expertos y otros con metodología algorítmica, lo que hace necesario la integración de diversas técnicas y estructuras de representación. Cuando la diversidad de técnicas es elevada y la flexibilidad de actuar con criterios diversos es un requisito, entonces los sistemas deben estar implementados bajo estructuras de control y representación no convencionales, es

decir bajo la filosofía de sistemas de comportamiento "inteligente". En este trabajo se hace una somera presentación de algunos de estos problemas y la utilización de estos criterios para la resolución de un sistema multiexperto dedicado a la interpretación de escenas de tipo industrial, fundamentalmente para su aplicación en tareas de ensamblaje (PIRS).

1 Introducción

La dificultad de integración entre diferentes tipos de representación y control en sistemas complejos hace necesario su resolución por métodos que aporten mayor flexibilidad que los convencionales. Sistemas para diagnosticar averias en redes telefónicas (Vesonder y otros, 1983) o para ayudar al diseño de circuitos VSLI (Kowalski y Thomas, 1984)o para diagnosticar enfermedades (Shortliffe, 1976), son algunos ejemplos. Entre los métodos que van adquiriendo mayor aceptació,n están los basados en técnicas de inteligencia artificial principalmente en el uso de sistemas multiexpertos. Estos sistemas presentan múltiples ventajas, pero su realización encuentra enormes dificultades cuando es necesario integrar técnicas y formas de representación.

En los sistemas complejos, pueden actuar a la vez diversos sistemas expertos que entre ellos solo intercambian alguna información, pero que disponen de datos y reglas diferentes, o bien otros procesos de tipo algorítmico. Asi por ejemplo, en el sistema de diagnóstico de redes telefónicas (ACE), el sistema experto se basa en otro de tipo convencional (CRAS) utilizado para adquirir datos y realizar los procesos previos. A partir de estos datos actua el experto sacando conclusiones y enviando en algunos casos información adicional a la base de datos común que posteriormente puede ser utilizada por ambos especialistas. En este caso, la complejidad e integración es reducida al ser solo necesario intercambiar datos a través de una única base de conocimientos. En otros casos, por ejemplo en el diseño de circuitos VSLI las bases de conocimientos son diversas y la complejidad aumenta.

Cuestiones tales como la representación y control, son temas que están intimamente ligados y que determinan los aspectos de complejidad e integración. En este artículo

decribiremos someramente algunos de aspectos y presentaremos la solución adoptada para un sistema multiexperto orientado a la interpretación de escenas de ensamblaje en robótica.

2 Sistemas multiexpertos : integración y complejidad

Los sistemas que trabajan en tareas interdisciplinarias, como por ejemplo en robótica, necesitan diversos especialistas que tienen varios modos de actuación, en linea o fuera de linea, y con estrategias de búsqueda diferentes , por ejemplo en anchura o profundidad. A menudo estos especialistas se basan en métodos conocidos que pueden ser formalizados mediante algoritmos, pero que precisan de una base de datos específica. Por otro lado, hay otros especialistas que no pueden ser formalizados, ya sea por la dificultad de formalizarlos o por el gran número de variables que intervienen, y normalmente son resueltos por medio de heurística. Estos especialistas son los candidatos para ser implementados por medio de sistemas expertos.

Si estos dos grupos de técnicas actuan sobre los mismos datos pero enfocados de forma diferente, lo más apropiado suele ser unificar todos los datos en una única base de datos, por ejemplo, en robótica entre especialistas de percepción y de planificación. Los primeros para tareas de reconocimiento o interpretación de escenas necesitan características concretas del objeto, es decir, el contorno o el nivel de gris, que sean accesibles de forma inmediata sin necesidad de un laborioso trabajo de deducción en la estructura geométrica y física. Por otro lado, el planificador de trayectorias tiene que razonar con la estructura geométrica, por ejemplo, rectas o planos, para encontrar la trayectoria en función de las restricciones del entorno y de los objetos. En este caso, el planificador precisa que el objeto esté descrito por planos o rectas y que se pueda acceder a ellos eficientemente.

En este ejemplo, los dos especialistas trabajan con un mismo objeto pero precisan información estructurada de forma diferente ya que los objetivos persiguidos son distintos. Hasta el momento, la mayoría de soluciones a este problema han estado dirigidas en la utilización de bases de datos diferentes fundamentalmente por motivos de tiempo

de cálculo y de espacio necesario. Aunque la solución es eficiente en casos sencillos, cuando se trabaja en tareas con mayor grado de complejidad, la eficiencia en algunos casos se convierte en complejidad en la mayoría, sobre todo por razones de integridad y flexibilidad. La unificación de la base de datos resuelve los problemas planteados, pero tiene como desventaja la necesidad de obtener la información de cada especialista y con ello se redunda en un aumento de tiempo de cálculo y espacio. La solución al problema es ardua y hasta el momento pocos sistemas la están tratando con toda su amplitud. Más adelante presentaremos un posible camino de solución utilizado en el sistema PIRS (Perception Interpretation Robotic System) (Sanfeliu, 1986).

Otro problema, es el control del flujo de información entre especialistas en sistemas complejos. Los sistemas conocidos de gobierno de un sistema multiexperto, (de tipo ascendente, descendente, heterárquico o de pizarra), son aplicables a unos u otros sistemas en función del tipo de problema y su complejidad. Por ejemplo, en sistemas de reconocimiento de formas, el gobierno general es de tipo ascendente porque a priori están perfectamente delimitadas las características y el dominio del problema. Asimismo, los sistemas de diagnóstico de enfermedades (Shortliffe, 1976) usando sistemas expertos, suelen ser de tipo ascendente porque parten de los síntomas para establecer las hípotesis del tipo de enfermedad. En cambio, otros sistemas del tipo de interpretación del lenguaje natural (Erman y otros, 1980),o de interpretación de escenas exteriores (Hanson y Riseman, 1978) hacen preciso el uso de estrategias generales de gobierno más flexibles. Asi en la interpretación del lenguaje general se han utilizado estructuras de pizarra, o en la interpretación de escenas exteriores estrategias heterárquicas. Actualmente la tendencia es mezclar algunas de estas dos últimas para resolver problemas concretos. El problema aparece en el momento de seleccionar una de ellas en sistemas multiexpertos. En la siguiente sección comentaremos más en detalle este apartado.

Finalmente está el problema de la validación de un sistema multiexperto puesto que a medida que se aumenta la complejidad resulta más dificil esta tarea. La experiencia obtenida a lo largo de los años, hace patente la necesidad de plantear los sistemas de la forma más modular posible y asi, poder aislar cada parte del sistema e investigarlo individualmente. Esto no es posible en algunos casos, pero al menos permite tratar los

sistemas dentro de márgenes razonables. Por ejemplo, en la interpretación de escenas del sistema ACRONYM (Brooks, 1981), cada una de las tareas es independiente del resto.

3 Tipos de representación y control general

Los tipos de representación más comunes de información simbólica en sistemas expertos pueden resumirse en : lógica de predicados de primer orden, sistemas basados en reglas, redes semánticas y marcos ("frames")(Barr and Feigenbaum, 1982). A estos hay que añadirles todos los de representación específica, por ejemplo, para describir la información geométrica de un objeto se puede usar la representación CSG (Constructive solid geometry)(Goos y Hartmanis, 1980). Dependiendo del tipo de problema hay representaciones más acordes que otras.

En lo que hace referencia a las representaciones de información simbólica, existen preferencias en función del problema al que se dirijan. Una de las representaciones más completas es la basada en la lógica de predicados de primer orden (Nilsson,1980), que presenta como ventaja su generalidad y "complitud" en cuando al razonamiento. Esta técnica de representación permite incluir información declarativa (fundamentalmente de "hechos") e información procedural mediante la inclusión de como proceder en el flujo (o control) del razonamiento. Además gran parte de las otras repesentaciones no son más que un caso particular de esta. La desventaja de esta representación reside en que su utilización en cuanto a razonamiento es, en general, co-NP-completo, es decir que tiene una complejidad en tiempo de orden exponencial. Debido a esta restricción, se ha reducido su utilización a problemas que requieran razonamiento lógico en una base de datos reducida o en base de datos híbridas.

Los sistemas basados en reglas no son más que una versión restringida de las basadas en lógica de predicados, pero que por su sencillez son enormemente útiles en problemas en donde el conocimiento sea de tipo declarativo, y en los que los datos o las reglas incorporan información imprecisa. Para ello separa la base de conocimientos del control, incluyendo en este último la metodología de razonamiento con incertidum-

bre. Esta representación es muy util cuando se dispone de información declarativa y el conocimiento no está estructurado, pero no es muy aconsejable, por ejemplo, en la descripción física de objetos. Por su enorme sencillez es esta una de las representaciones más usadas, aunque normalmente su estructura práctica no suele ser pura, sino más bién un sistema híbrido entre sistema de producciones y sistema secuencial.

Las redes semánticas surgieron del estudio de memorias asociativas y ha venido a ser una de las representaciones más potentes que se disponen por varias razones, entre las que se encuentra la facilidad de interconectar información conceptual a diferentes niveles y poder razonar a través de ella, la incorporación del tiempo (Brachman y Levesque, 1985), y los atributos específicos en cada concepto. Además las redes semánticas permiten agrupar grupos de conceptos que describen declarativamente el espacio de los cuantificadores universales. Las redes semánticas son fundamentalmente apropiadas para describir conceptos que esten relacionados, por ejemplo la descripción de navios o la descripción del sistema cardiovascular.

Los marcos pueden considerarse como un caso específico de redes semánticas, aunque algunos autores prefieren distinguir ambas representaciones. En un principio los marcos (frames) agrupan información de un concepto a través de la descripción de sus partes, y se conectan a otros marcos por medio de los registros (slots). En sí la representación global es de tipo de red semántica aunque las formas de razonamiento difieren. Los marcos son muy útiles para describir estructuras o conceptos relacionados.

Cada una de estas representaciones está mejor orientada a ciertos tipos de representación del conocimiento, sobre todo en cuanto al razonamiento a través del conocimiento. Esto lleva consigo que en sistemas multiexpertos no sea factible utilizar una única base de representación del conocimiento y sea necesario aprovechar las ventajas de algunas de estas representaciones, aunque siempre tratando de minimizar las representaciones para reducir los problemas de coherencia e integridad de los datos.

El control global de un sistema es otro de los elementos básicos en la estructura de un sistema multiexperto, porque de él depende como se va a gobernar el flujo de información del sistema. Existen en la actualidad, fundamentalmente cuatro tipos de control global denominados ascendente, descendente, heterárquico y de pizarra. El ascendente o guiado por los datos, está orientado a tareas en las que se parte de la

hipótesis que su resolución puede ser conseguida de una sola vez a partir de los datos que se disponen. Asi por ejemplo, en problemas de diagnóstico automático, si se puede relacionar univocamente los síntomas con la enfermedad, un sistema con control ascendente es suficiente para diagnosticar la enfermedad. El problema reside en que amenudo no existe una relación unívoca entre los síntomas y la enfermedad, ya sea por que solo se dispone de información parcial o por la imprecisión de los datos.

El control de tipo descendente parte de los objetivos y trata de imponerlos en los datos, tanto sea para verificar su existencia como para guiar y obtener la información necesaria. Por ejemplo en problemas de ayuda son de enorme utilidad, asi en la concesión de créditos personales, el sistema puede guiar al cliente por medio de preguntas que no son más que los datos que precisa para llegar a la meta. Este tipo de control es útil cuando se quiere guiar la búsqueda a partir de los datos del objetivo,asi como en sistemas de verificación de datos.

El control de tipo heterárquico es una combinación de un control de tipo ascendente y otro descendente, conjugando las ventajas de ambos métodos. Es decir, una parte del sistema actua de forma ascendente y la otra descendente, aunque en algunos casos pueden estar integradas ambas estructuras. Este tipo de control es uno de los más usados en los sistemas actuales debido a que en la mayoría de ellos hay partes conocidas a priori y que pueden ser gobernadas a partir de los datos y otras a partir de los objetivos. Un ejemplo de estas últimas, son los sistemas de visión por computador donde el paradigma de "hipótesis-verificación" utilizado para reconocer e interpretar una escena, se amolda adecuadamente a esta estructura de control.

Siempre y cuando la estrategia de resolución de un sistema se conozca a priori y no sea necesario modificarla en el funcionamiento del sistema, los tipos de control global descritos son suficientes para gobernar un sistema de cualquier tipo de complejidad conjugando además la máxima eficiencia del sistema. Ahora bién, si el sistema necesita cambiar su estrategia de funcionamiento en función de los resultados obtenidos por especialistas del sistema, por ejemplo en la interpretación del habla, entonces es más adecuado utilizar una estructura de control de pizarra. Este tipo de control se basa en el principio, que diversos especialistas actuan independientemente sobre el mismo problema y plantean su resolución al coordinador general. Éste, en función de los

resultados de cada especialista decide quién y cuando tiene que actuar sobre el sistema. Por ejemplo en un sistema de visión por computador, diversos especialistas en el reconocimiento de escenas (el de reconocimiento de árboles, o el de carreteras para escenas al exterior), pueden informar al coordinador de sus interpretaciones parciales y este decide quién tiene que continuar la búsqueda. Esta estructura de control es muy útil en sistemas de interpretación, aunque presenta el inconveniente del enorme tiempo de proceso, ya que cada especialista actua de forma independiente.

4 El sistema PIRS

Un sistema multiexperto dedicado a la interpretación de escenas 3D a partir de varias fuentes de percepción (denominado PIRS), que se está construyendo en el Instituto de Cibernética, plantea varios de los problemas generales de integración y complejidad enumerados previamente. Los objetivos del sistema se pueden resumir en :

- Identificar los objetos de una escena de tipo industrial.
- Verificar los cambios acontecidos en una escena en tareas de supervisión.
- Identificar un objeto dentro de la escena.
- Adquirir (aprender) nuevas piezas por medio de visión por computador y de un sistema CAD.
- Predecir y razonar con la información de diversos sensores.

En la actualidad se está trabajando en el primero de estos objetivos.

4.1 Descripción

El sistema se basa en varios paradigmas, el de "hipótesis-verificación", el de "concentrar la búsqueda" y el de "predecir información", todos orientados a controlar y reducir la búsqueda en la escena. El primer paradigma es de nivel general en cuanto no se

dispone de información a priori de los objetos subyacentes en la escena. Su forma de funcionamiento consiste en emitir hipótesis sobre los posibles objetos que intervienen en la escena después de las etapas de adquisición, tratamiento y obtención de características, y verificarlos hasta interpretar la escena. Esta estrategia general ha sido utilizada previamente por otros autores en sistemas de interpretación de escenas entre los que destaca el ACRONYM (Brooks, 1980), y el VISIONS (Hanson y Riseman, 1978).

El paradigma de "concentrar la búsqueda" es de tipo local en cuanto que se restringe su actuación a las etapas de verificación y predicción, y su meta es reducir la búsqueda concentrándose unicamente en aquellos aspectos que lo requieran. Este paradigma es también muy conocido y cada vez más usado en los sistemas de interpretación para eliminar la búsqueda explosiva. El tercer paradigma, "predecir información" tiene caracter general y está orientado como el predecesor a reducir la búsqueda cuando se dispone de información a priori o cuando no se tienen hipótesis válidas de interpretación. Este paradigma se ha usado en pocos modelos de interpretación, por ejemplo en el ACRONYM, pero con una estructura restringida.

El esquema genérico de funcionamiento del PIRS, puede verse en la Fig.1, en donde se detallan diversos componentes de las dos etapas diferenciadas: la de obtención de datos de la escena (guiada por los datos) y la de interpretación. Su estructura de control es de tipo mixto heterárquico y de pizarra, la primera para el conjunto de actuación del sistema y la segunda para controlar la interpretación. La etapa guiada por los datos está constituida por los procesos que se realizan sobre la escena hasta extraer la información básica de los objetos y de la escena. En esta etapa intervienen entre otros, procesos de filtrado, detección de bordes, obtención de regiones, extracción del contorno exterior y obtención de características primarias que servirán en la elaboración de las hipótesis iniciales. La obtención de características puede ser de dos tipos: una genérica en donde se extraen características fijas, y otra específica que depende de las peticiones de la etapa de intepretación.

A partir de estos datos se entra en la etapa de intepretación, en donde pueden intervenir los módulos de clasificación, verificación y predicción. En función de estos y de los extraidos en la etapa guiada por los datos, se realizan las hipótesis iniciales

(los objetos candidatos que están en la escena) y para cada una de ellas se obtiene la posición y orientación de cada uno de los modelos reconocidos. Si el proceso ya está en fase de interpretación se puede obtener de la primera etapa unicamente información concerniente a un aspecto concreto, por ejemplo la obtención de una arista en una zona de la escena.

.Ya que de todas las hipótesis solo algunas pueden ser ciertas, la fase de clasificación inicia una serie de procesos con un doble objetivo. Por un lado, averigua el grado de certeza de que el objeto pueda ser alguno de los modelos seleccionados, y por otro, extrae información de lo que no se ha encontrado en la escena pero que debería estar para que fuese el modelo seleccionado. Con la información suministrada por el clasificador se pasa a la fase de verificación en donde se trata de completar la información del objeto en lo que respecta a la clasificación, es decir se busca a través de las características obtenidas o examinando de nuevo la escena aquellos datos que faltan. Por ejemplo, si en la imagen se ha perdido una arista, a través del conocimiento de los planos que la generan puede comprobarse su existencia. En caso que ninguna de las hipótesis haya obtenido el grado de certeza suficiente para poder pasar esta fase, entonces debe actuar el predictor para obtener las hipótesis pausibles.

El módulo de verificación está basado en una serie de razonadores orientados a inferir información de la estructura (razonador geométrico), de la propiedades físicas (razonador simbólico y sobre las características físicas), de las relaciones objeto-entorno y de las relación objeto-dominio (razonador simbólico). Una de las aplicaciones de este último son en tareas de supervisión por ejemplo, en donde es más importante encontrar alguna característica determinante que tratar de razonar sobre la estructura geométrica del objeto. En la Fig. 2, se muestran algunos de los razonadores y planificadores utilizados en esta fase, en la que los planificadores tienen por misión, reducir y ordenar la búsqueda, y redistribuir las tareas en función de los recursos disponibles y de las estrategias generales consideradas.

El módulo de predicción (predictor) está detinado reducir el tiempo de búsqueda en una escena cuando se conoce a priori los objetos que la componen, alguna característica destacable o relaciones objeto-entorno o objeto-dominio. En este caso, el predictor trata de hallar los elementos necesarios para reducir la búsqueda, razonando con la estructura

geométrica, física del objeto y con la simbólica (etapas similares a la del verificador), y teniendo en cuenta los recursos disponibles y las estrategias generales consideradas. La información del predictor es por ejemplo, la forma de guiar la segmentación, que características hay que tomar en cuenta en la clasificación y como hay que realizar la verificación. Todo ello permite a priori gobernar la extracción y reconocimiento de los objetos de una escena.

Estas tres fases están gobernadas por el interpretador a través de un sistema multi-experto que explicaremos a continuación.

4.2 Representación y control

El sistema PIRS planteó la necesidad de tomar en cuenta un sistema de representación y control que permitiera tener diferentes tipos de datos y de técnicas. Por ejemplo, en el sistema se agrupan tanto técnicas algorítmicas como no algorítmicas, es decir técnicas de clasificación puramente basadas en la metodologia de reconocimiento de formas y técnicas de sistemas expertos basadas en metodologia de inteligencia artificial (en la Fig. 3 se muestran las funciones del interpretador en relación a la base de conocimientos). Por lo que respecta a representación, tanto hay datos de tipo geométrico como las reglas necesarias que regulan las relaciones objeto-dominio.

Haciendo balance de la hetereogeniedad del sistema y tomando en cuenta lo expuesto en el apartado 3, se ha diseñado el sistema con estrategia general de estructura heterárquica y dentro del interpretador con estructura mixta, heterárquica y de pizarra (Hayes-Roth, 1985). La estrategia heterárquica viene motivada por la problemática propia de los sistemas de visión. Por un lado es necesario poder actuar sin información a priori, es decir, a partir de los datos de la imagen hay que reconocer los objetos que la componen utilizando unicamente los datos de la propia imagen, lo que es un proceso tipicamente ascendente. Por otro lado, una vez se dispone de esta información, hay que analizarla a partir de los modelos de referencia. Esta tarea puede hacerse de forma ascendente como un proceso de reconocimiento de formas, siempre y cuando se conozca en profundidad toda la información de la escenas y las relaciones entorno-objeto y dominio-objeto, pero en general para sistemas de gran flexibilidad

como el PIRS, no se parte de dicha información con lo que esta tarea debe realizarse de forma descendente y con estructura en pizarra.

La estructura mixta del interpretador es consecuencia de la interacción entre los tres módulos, el clasificador, el verificador y el predictor. Según el tipo de información que se disponga en la escena, el sistema estará más o menos guiado por el interpretador, es decir si de la escena no se dispone de información a priori y las hipótesis iniciales avalan una interpretación rápida, entonces el sistema actuará de forma descendente guiada por el interpretador. En caso contrario, si bién se dispone de información a priori o las hipótesis iniciales no son suficientemente clarificadoras, entonces cada uno de los módulos actuará de forma independiente, indicando al interpretador las acciones que tomarían en cada caso, siendo este último el árbitro y conductor de la secuencia de operaciones a realizar.

Existen diferentes formas de comunicar estos procesos en lo que se refiere a las bases de datos intermedias. Amenudo, esta interrelación se hace a través de varias bases de datos intermedias, cada una con su propia estructura, pero esta técnica presenta problemas en cuanto es necesario que el interpretador tenga acceso a gran parte de las hipótesis que se vayan generando. Por ello se ha construido una estructura en agendas que presenta diversas ventajas (Hayes-Roth, 1985). La estructura en agendas puede verse en la Fig. 5. Cada una de estas agendas contiene mensajes codificados según :

$$Numero/Tipo-de-mensaje/Estado/Datos$$

El "Número" es una cifra correlativa que identifica a un mensaje, el "Tipo-de-mensaje" indica como hay que leer la información que dispone este mensaje, el "Estado", en que situación se encuentra esta información (si ha sido o no utilizada, etc...) y los "Datos", la información propia del mensaje. Los mensajes son leidos y grabados por cada uno de los procesos principales. Asi por ejemplo, una vez se han extraido las características de la imagen (las regiones, el contorno exterior, la descripción por rectas, etc...) estas son enviadas a una agenda en forma de mensaje. Esta estructura de agenda facilita enormemente las tareas de comunicación entre procesos y da flexibilidad al sistema.

El interpretador es el módulo principal que controla a los demás procesos, el clasificador, el verificador y el predictor. Para ello cuenta con una estructura de sistema de producciones, combinando conocimientos propios de gobierno y de interpretación. En él están las estrategias generales de interpretación, la guia de los procesos dentro de cada una de las estrategias, y la forma de como conjugar los diferentes recursos. El interpretador es en sí, un sistema experto que activa a los procesos de clasificación, verificación y predicción, y que decide la actuación de ellos en función del estado de la interpretación.

El clasificador realiza dos procesos uno el de obtención de las hipótesis primarias y otro de obtención de la medida de similitud (Sanfeliu, 1984) entre cada objeto de la imagen y los modelos seleccionados inicialmente. Ambos procesos tienen estructura algorítmica. El analizador tiene una doble misión, por un lado calcula la medida de similitud entre el objeto encontrado y el modelo, y por otro envia a la agenda la parte de la estructura que no ha encontrado en el análisis. Por ejemplo, si falta una superficie, el clasificador envia a la agenda información sobre esta superficie y a su vez los valores de clasificación. Este módulo está estructurado en forma algorítmica, y hace uso de información orientada al reconocimiento de objetos.

El verificador se basa en una estructura diferente. Su base de control es un sistema de producciones que gobierna los subprocesos de razonamiento y planificación a partir de la información de la agenda y de la guia del interpretador. Los subprocesos de razonamiento tienen estructuras diversas: el razonador geométrico presenta una estructura de sistema de producciones y su razonamiento es de tipo algebraico; el razonador geométrico-físico, por un lado, presenta estructura de sistema de producciones y por otra de red semántica en donde está toda la información concerniente al objeto; y el razonador simbólico, que es el que tiene la misión de razonar con el objeto-entorno , el objeto-dominio y los objeto-sensores, presenta una estructura de sistema de producciones. Cada razonador tiene integrado un planificador que reduce y ordena la búsqueda. El predictor se basa en una estructura similar a la del verificador.

La base de conocimientos del sistema global está constituida por diversos tipos de representación debido a la diversidad de información necesaria (Fig.4). Los objetos están descritos por un lado por una base de datos de "fronteras" (Goos y Hartmanis,

1980), destinada al diseño de la pieza y a su representación, y por otro por una red semántica (Fig.6) en donde se relacionan las propiedades geométricas con las físicas, así como las escenas con los objetos. Esta red semántica tiene sus nodos definidos como marcos en los que en los registros están las propiedades geométricas y físicas. Las relaciones entre los nodos son de diversos tipos como por ejemplo "Parte de". Dentro de esta red, las relaciones entre superficies están descritas por gramáticas arbol-grafo (Sanfeliu y Fu, 1982), lo que permite realizar la medida de similitud entre el objeto y el modelo según el algoritmo descrito en (Sanfeliu, 1984).

Las relaciones objeto-entorno, objeto-dominio, de los sensores y de las estrategias, están representados según representaciones de sistema de producciones, habiendo diferentes bases de datos. La información de los sensores no solo describe las características entre ellos sino que además va a permitir razonar en la estructura del objeto según el sensor que se utilice, es decir incorpora información de que reglas deben aplicarse para transformar los objetos en las partes percibidas. Este último punto es de gran utilidad para que el predictor y el verificador puedan dirigir la búsqueda sin tener que hacer transformaciones posteriores y sin tener que basarse en una base de datos específica.

5 Conclusiones

En este trabajo se presentan dos problemas importantes en cuanto a la realización de sistemas multiexpertos, la representación y el control, que determinan su complejidad y dificultad de integración. La representación es uno de los problemas básicos en cuanto a la eficiencia de razonamiento y funcionamiento, la flexibilidad en modificar y estructurar información, y suele ser uno de los elementos esenciales en lo que se refiere a la versatilidad del sistema en diferentes entornos y dominios de aplicación. Por otro lado, el control es el elemento que determina el funcionamiento del sistema y la base de su comportamiento "inteligente", es decir que permita resolver situaciones no previstas inicialmente.

En este trabajo se presenta la resolución de un sistema multiexperto orientado a la interpretación de escenas a partir de sensores. En él se abordan los problemas de representación por medio de la elección de dos tipos de estructuras de tipo simbólico,

una red semántica y una basada en reglas, y una de tipo geométrico (de fronteras) para representar los objetos. Estas representaciones se conjugan gracias a la utilización de especialistas que permiten obtener información de cada una de ellas. En lo que se refiere al control se ha elegido un estructura híbrida que combina una heterárquica y otra de pizarra. La primera se usa para gobernar el conjunto, y entre las dos para gobernar al interpretador y a sus módulos integrantes. Esta combinación va a permitir dotar al sistema de un alto grado de versatilidad.

6 Referencias

Barr A. and Feigenbaum E.A. The handbook of artificial intelligence. *Pitman Books Limited, 1982.*

Brachman R.J. and Levesque H.J. Readings in knowlegde representation. *Morgan Kaufmann Publ., Inc., Los Altos, 1985.*

Brooks R.A. Symbolic reasoning among 3D models and 2D images. *In Computer Vision (J.M.Brady ed.), North Holland, Amsterdam, New York, 1981.*

Erman L.D., Hayes-Roth F., Lesser V.R. and Reddy D.R. The Hearsay-II speech understanding system: Integrating knowlegde to resolve uncertainty. *Computing Surveys 12, 1980, pp.213-253.*

Goos G. and Hartmanis J. Computer aided design modelling, systems engineering, CAD-systems. *In Lecture Notes in Computer Science n. 89, Springer-Verlag, New York, 1980.*

Hanson A.R. and Riseman E.M Segmentation of natural scenes. *In Computer Vision Systems (A.R. Hanson and E.M. Riseman), Academic Press, New York, 1978.*

Hayes-Roth B. A blackboard architecture for control. *Artificial Intelligence 26, 1985, pp.251-321.*

Kowalski T.J. and Thomas D.E. The VSLI design automation assistant: an IBM system/370 design. *IEEE Design and Test, vol.1, n. 1, Feb. 1984.*

Sanfeliu A. A distance measure based on tree-graph-grammars: A way of recognizing hidden and deformed 3D complex objects. *7th Int. Conference on Pattern Recognition, Montreal, Canada, July 30-Aug. 2, 1984.*

Sanfeliu A. PIRS: Sistema de percepción interpretación para robótica. *Informe interno Instituto de Cibernética, 1986.*

Sanfeliu A. and Fu K.S. Tree graph grammars for pattern recognition. *In Lecture Notes in Computer Science n. 153, Graph-Grammars and Their Application to Computer Science (Ehrig H, Nagl M., and Rozenberg G. eds.) Springer-Verlag, New York, 1983,pp. 349-368.*

Shortliffe E.H. Computer-based medical consultations: MYCIN. *Elsevier-North Holland, New York, 1976.*

Vesonder G.T., Stolfo S.J., Zielinski J.E., Miller F.D. and Copp D.H. ACE: an expert system for telephone cable maintenance. *Proc. 8th IJCAI, 1983, pp.116-121.*

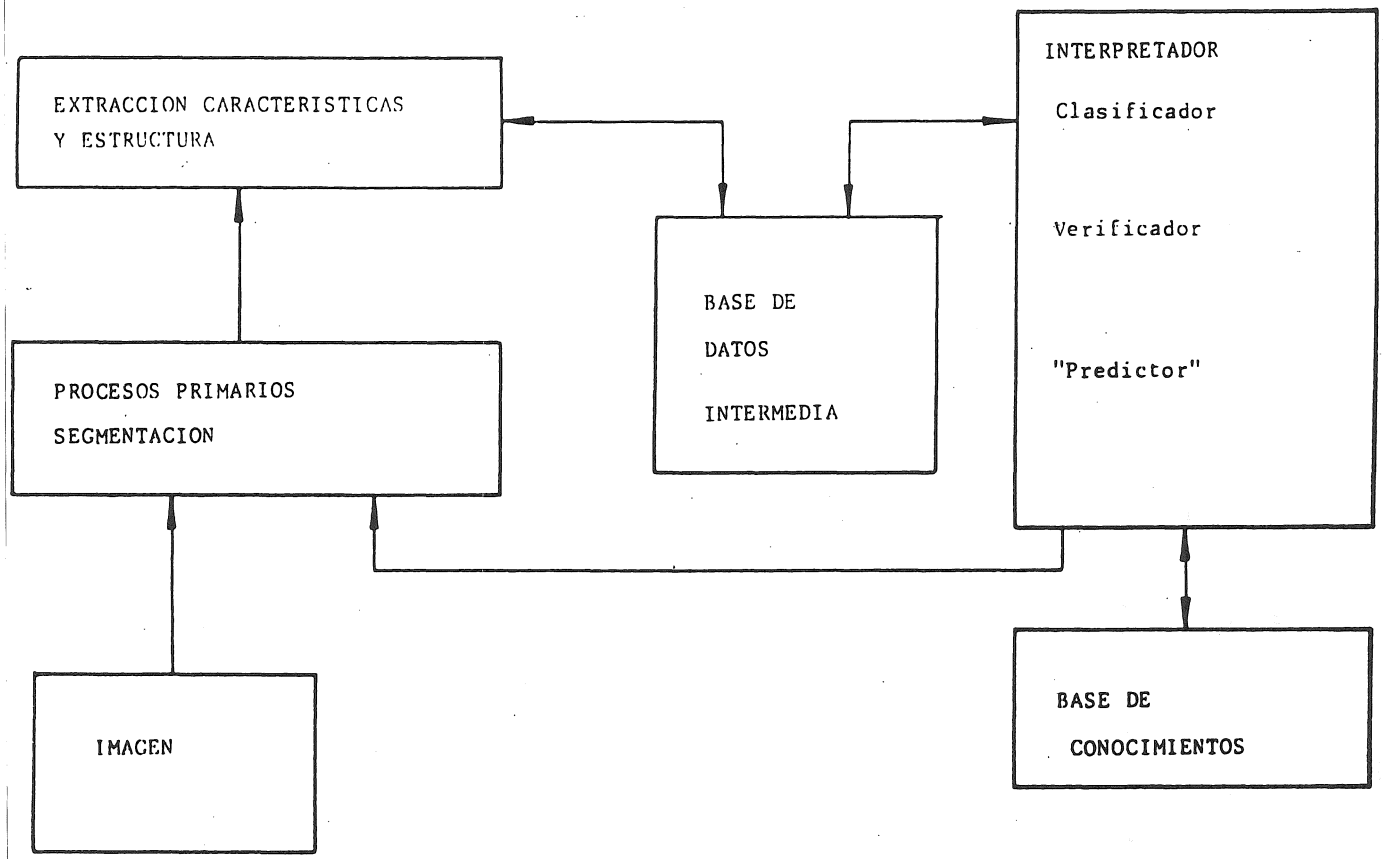


Fig. 1 - Diagrama global del sistema PIRS

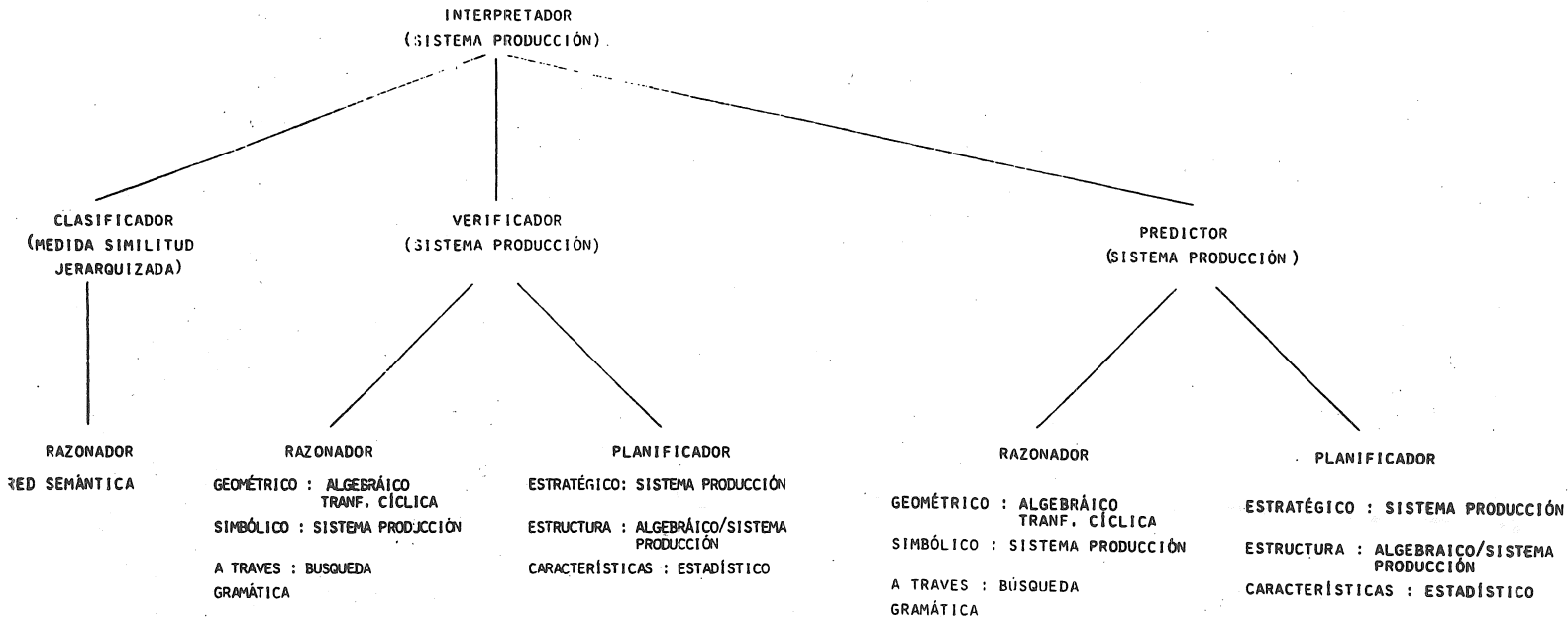


FIG. 2 - TECNICAS UTILIZADAS EN CADA MÓDULO

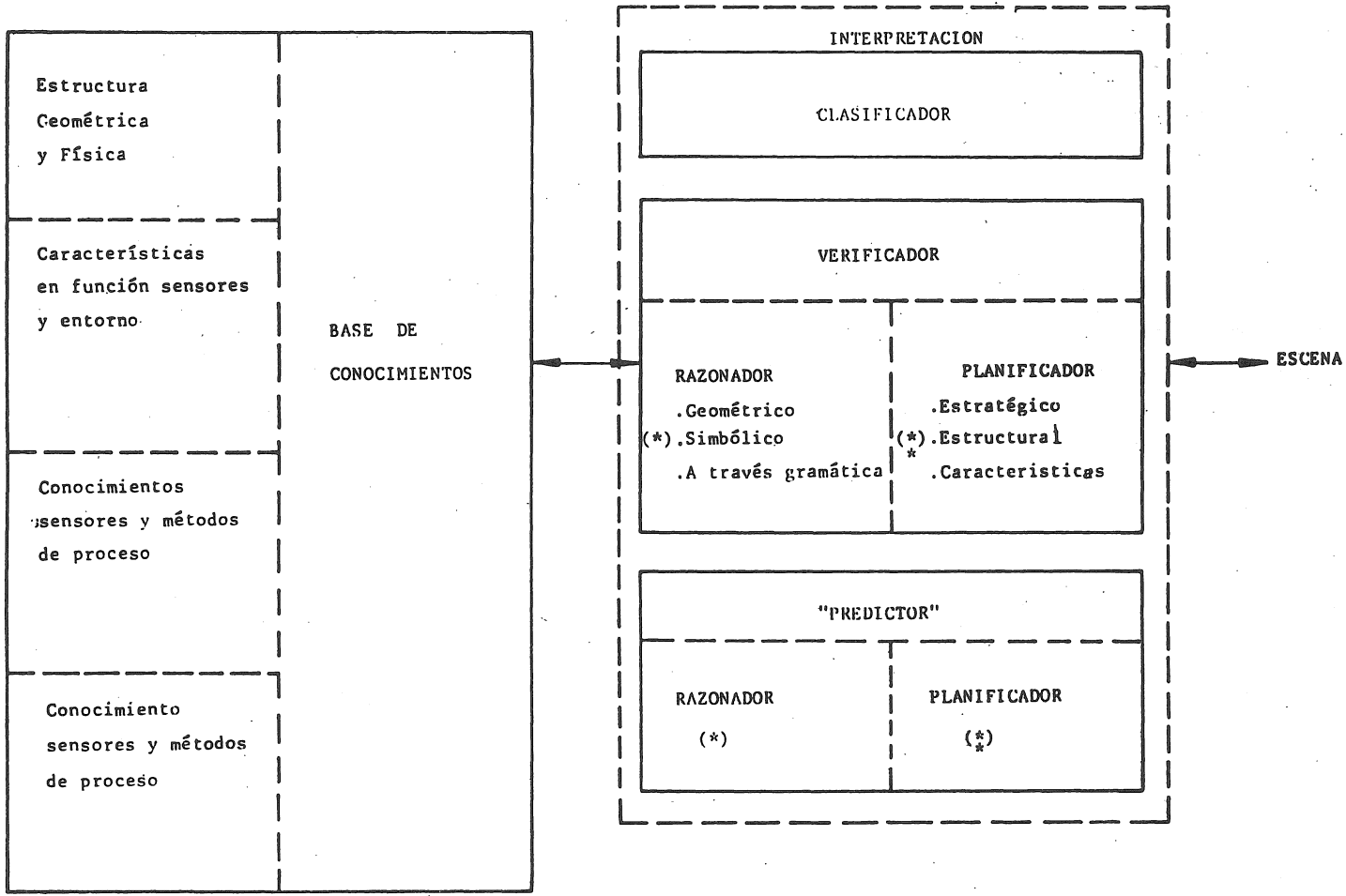


Fig. 3 - Conjunto interpretador - base de conocimientos

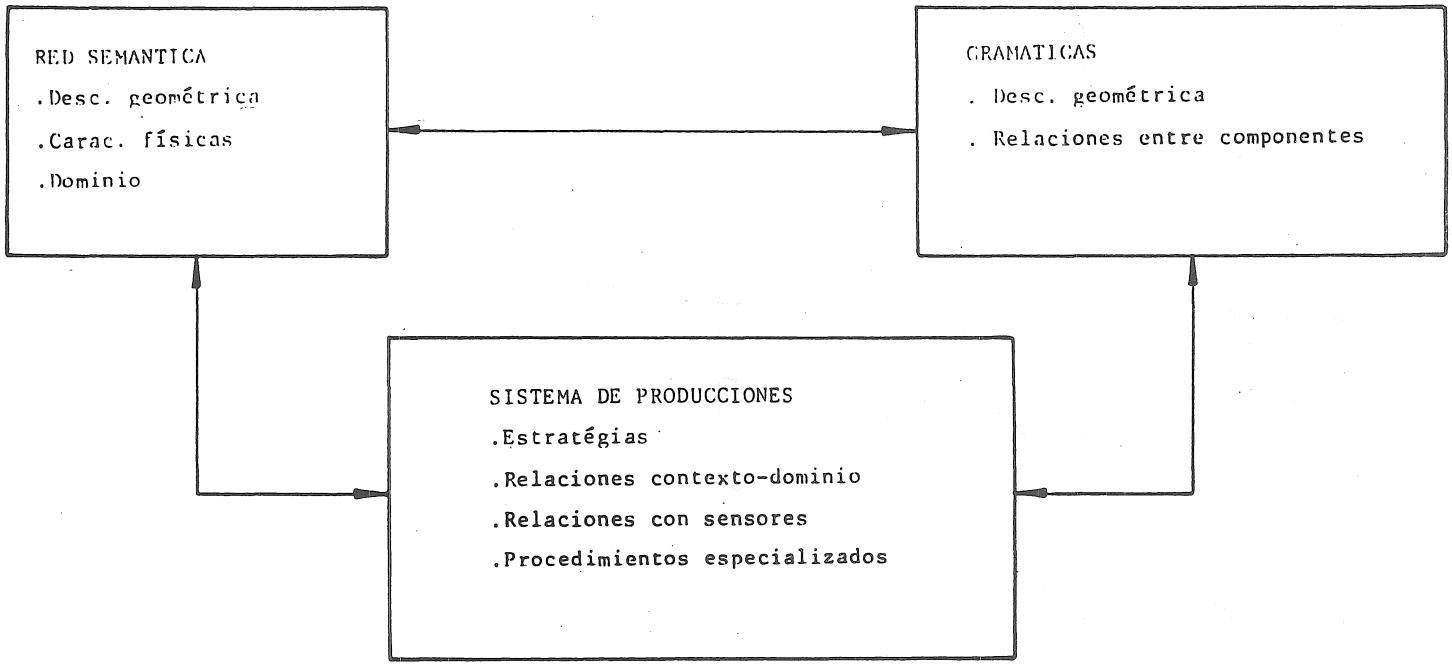


Fig. 4 - Diversas interpretaciones usadas en la base de conocimientos

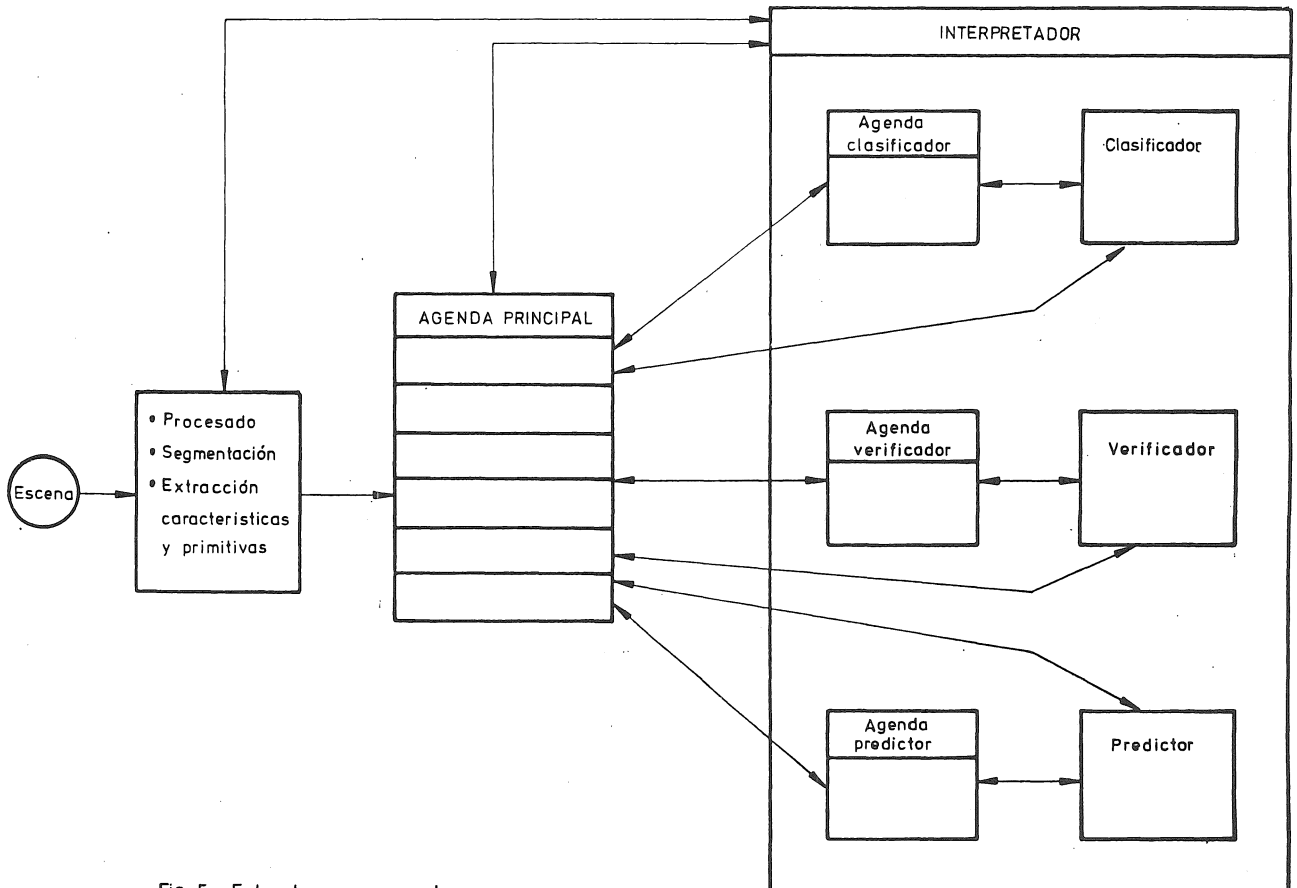


Fig. 5. Extructura en agendas

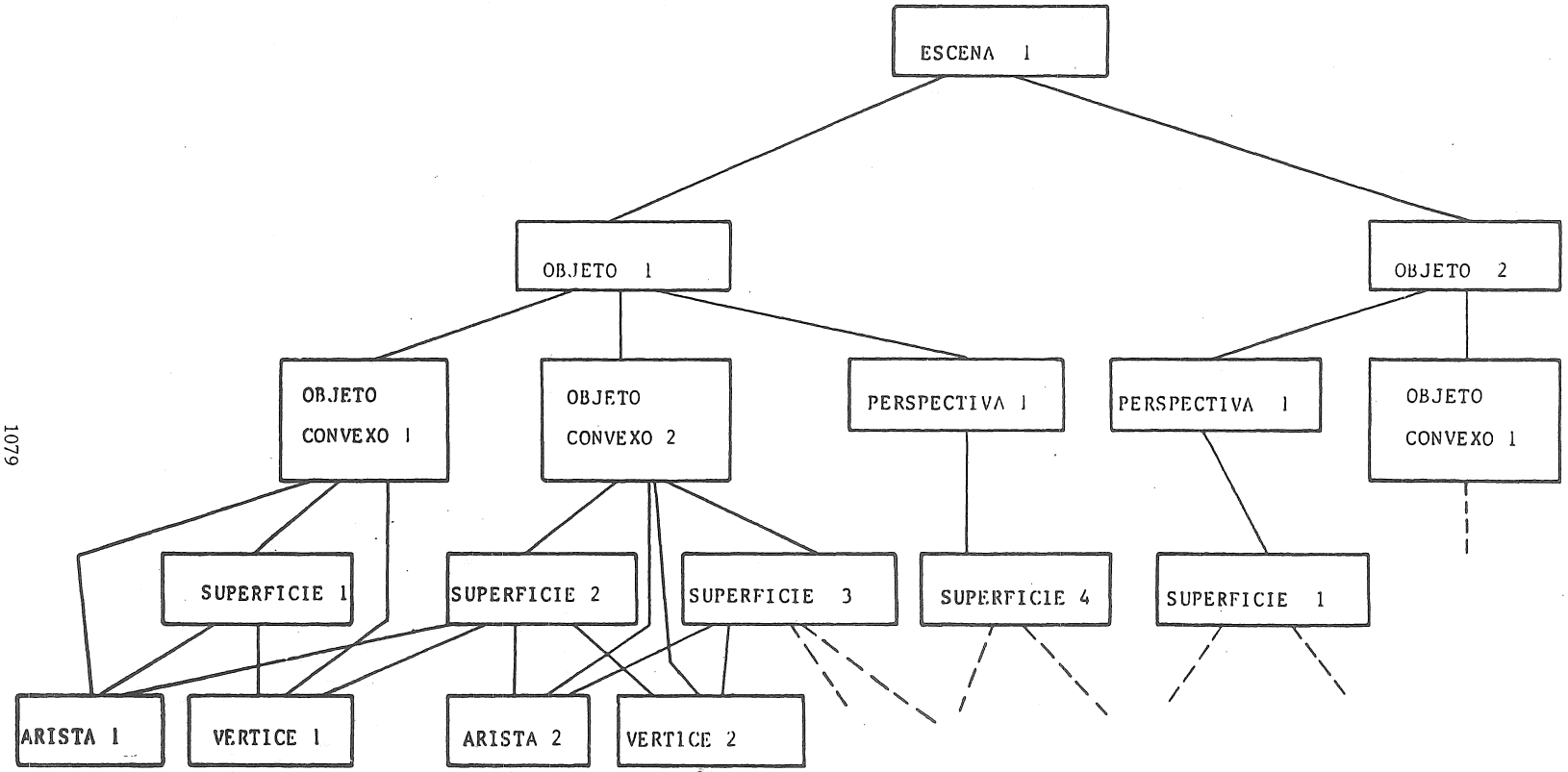


Fig. 6 - Vista parcial de la red semántica

REVISTAS EN HIPERMEDIA. MODELOS Y EXPERIENCIAS

C.C. Alejandra Schiavoni, Lic. Claudia Queiruga, Lic. Marcelo Hourqueby, Lic. Silvia Dómini, Sr. Fernando Das Neves.
LIFIA, Facultad de Ciencias Exactas, Universidad Nacional de La Plata. Calle 50 y 115, Primer Piso, C.C. 172.
E-mail: jdiaz@unlp.edu.ar
Fax: 54 21 25 7240.

ABSTRACT

Este artículo se centrará, básicamente, en cómo es una revista presentada en un medio electrónico usando la técnica de hipermedia. Primeramente, se hace un análisis de "la revista", contraponiendo las ventajas que ofrece la versión electrónica con las restricciones de la producción en papel. Finalmente, se hace una breve descripción de dos implementaciones existentes que se basan en los mismos conceptos y que se aplican a dos dominios de interés diferentes.

KEYWORDS

Revista electrónica, hipermedia, recuperación de información, metáfora, navegación.

¿PORQUE UNA REVISTA ELECTRONICA EN HIPERMEDIA?

Una revista convencional tiene como principal ventaja su portabilidad, esto es, el lector puede leerla en cualquier momento y lugar; característica ésta, díficil de lograr con una revista electrónica, aunque existe la posibilidad de utilizar pen-tops o notebooks, que apuntan a incrementar la portabilidad de la misma. Una revista electrónica es una alternativa diferente de presentación de la información y no un reemplazo de la clásica revista de papel.

Una revista en papel tiene una organización secuencial que no se corresponde con la naturaleza asociativa de la mente humana ni con la manera de exploración o investigación que un lector realiza cuando lee un artículo de su interés. A continuación, se describirán algunas deficiencias observadas en el medio de papel, las que son superadas usando un medio electrónico: 1) Es tedioso para el lector

encontrar la información de su interés dentro de un bloque extenso de texto; 2) Es difícil obtener la información en una secuencia distinta a la determinada por el autor; 3) Es imposible presentar apropiadamente diferentes dominios de conocimiento, debido a que existen relaciones que no se pueden representar en una estructura lineal; 4) Es difícil integrar y actualizar grandes volúmenes de información frecuentemente cambiante y proveniente de distintas fuentes; y 5) El estilo de organización de un libro se caracteriza por la ineficiencia en cuanto al almacenamiento y recuperación de la información, ya que los mismos contenidos aparecen repetidamente en diferentes puntos.

Sin embargo, estas limitaciones no corresponden al medio, sino que las fuerza el soporte: el papel. Como tantas otras tecnologías, la herramienta y la forma de uso se entrecruzan de una manera que las hace parecer obvias y únicas. Es el papel y no la revista como concepto quien obliga a esta organización. Pero el papel está por perder su exclusividad. El avance de la tecnología en informática ha hecho que los medios de transmitir información (tanto físicos como conceptuales) experimenten cambios . Es posible acercarse más, mucho más, a la esencia de la información adaptando los recursos sobre los que se expresa y no forzándola al formato del medio. La revista electrónica es uno de estos nuevos medios. Hipermedia es la tecnología.

Una revista electrónica sobre una base de hipermedia puede parecerse mucho o muy poco a la revista tradicional. Puede (como se explica más abajo) tras una fachada conocida hacer disponibles nuevos medios de navegación, nuevas operaciones que potencian la capacidad natural de asociación del pensamiento. Operaciones como: 1) Posibilitar un acceso rápido a un punto específico; 2) Buscar una

idea sin saber su forma o expresión exacta; 3) Vincular conceptos que están en artículos diferentes de manera de compararlos, o seguirlos, o saltar de uno a otro (Nótese que si bien se habla de artículos, el límite preciso que ésto implica se hace un tanto borroso al representarlos en hipertexto/hipermedia); 4) Organizar los artículos en formas no convencionales (creando niveles de lectura que se complementan, o haciendo que un reportaje se transforme en un hecho activo aún para el lector, dejando que seleccione la próxima pregunta); y 5) Aprovechando una consecuencia del modelo de hipermedia, como es la imprecisión de la frontera entre lector y autor, permitir que el lector agregue comentarios, expanda el texto o agregue links, ambos al mismo nivel que los hechos por el autor.

Según se ve, las posibilidades de hipermedia operan sobre un concepto básico: transformar lo que previamente eran estructuras y actores pasivos en activos.

Un lector, rara vez lee artículos de una revista en forma secuencial o lineal, ya que su misma motivación lo induce a buscar sólo la información de su interés, dejando de lado aquella que en ese momento es tangencial a su inquietud. Justamente esta manera de explorar la información, es la propuesta por los diseños basados en hipermedia.

La técnica de hipermedia aplicada en el diseño e implementación de una revista electrónica, permite disgregar los artículos contenidos en ella en conceptos independientes relacionados entre sí a través de "links", que vinculan los conceptos por la semántica de su contenido. Organizando la información de esta manera el lector tiene la posibilidad de tomar sus propias decisiones de acuerdo a su interés, cuando está leyendo un artículo, obteniendo así el enfoque del tema que él desea. Además, la posibilidad de utilizar información multimedial, le permite al autor transmitir ideas que

a través del papel le resultarían casi imposibles, por ejemplo una animación podría explicar un fenómeno de la naturaleza en forma más clara y didáctica que un texto chato o un simple dibujo. Desde el punto de vista del lector, el uso de multimedios induce a una mayor compenetración en el tema que se está tratando, facilitando la comprensión del mismo. Por otro lado, motiva al lector a mantener su atención en el tema.

A continuación, haremos una breve descripción de la aplicación de las ideas anteriormente expuestas, en la implementación de dos aplicaciones: una revista electrónica para la enseñanza de geografía Argentina ("Conociendo el País"), y la traslación de una revista previamente existente ("David y Goliath") del Consejo LAtinoamericano de Ciencias SOciales.

DESCRIPCION DE LOS PROTOTIPOS

"Conociendo el País"

Objetivo

El objetivo es construir una revista electrónica en hipermedia, orientada a niños, destinada a la enseñanza de la Geografía Argentina.

La información contenida en ella está organizada como un conjunto de artículos que abordan diferentes temas. La idea es generar una biblioteca donde se irán guardando los distintos números, permitiendo al lector seleccionar el número que desea leer.

Por tratarse de una revista destinada a niños la interfase se caracteriza por la inclusión de gráficos y animaciones que motivan al lector y brindan una visión integrada del tema sobre el que se está informando.

Descripción detallada:

La revista consta de: 1) Una tapa que representa el comienzo de una historia, en donde se relata el viaje de un niño explorador que

invita al lector a recorrer lugares turísticos del país; 2) Un índice que simula el tablero de control de un aeropuerto que anuncia los posibles destinos a los que se puede arribar y 3) Los artículos que componen el contenido de la revista y un anticipo con las notas periodísticas del próximo número. Los artículos constan de notas referentes a puntos atractivos del país y un circuito turístico, diferente en cada número, que destaca los lugares representativos que pueden visitarse en algunas de las ciudades más importantes.

En el diseño de la red de información se trató de evitar la desorientación del lector durante la navegación o exploración de la red, proveyendo mapas que muestran todo el grafo resaltando el nodo actual y el camino recorrido hasta el momento. Esta pista provista por el modelo permite que el lector sea capaz de responder a preguntas tales como: Dónde estoy ubicado dentro de la red?, Por dónde puedo continuar?, Qué he visto?, facilitándole cualquier toma de decisión en cuanto a la forma de continuar su lectura.

Para atacar el problema de que el lector obtenga más información de la que puede asimilar, también conocido como "overhead cognitivo", se puso énfasis en dos puntos relevantes dentro del diseño: 1) En cuanto a la interfase, se mantuvo un diseño coherente y una simbología uniforme y, 2) Con respecto a la organización de la información se trató de evitar que a partir de un nodo, un lector puede internarse demasiado en la red, resultándole imposible retornar al punto del cual partió.

"David y Goliath"

Descripción Detallada:

La idea gira alrededor de disminuir la dificultad del paso entre medios. El diseño de la revista se podría dividir en dos campos: El del acceso a los contenidos y el de la representación de los

contenidos en sí. Si bien un y otro implican técnicas no necesariamente relacionadas, el nexo de comunicación entre ambas surgió mediante la búsqueda de metáforas que uniera y diera coherencia a acciones y contenidos (entendiendo metáfora como la representación no literal de objetos, en la cual se adopta un modelo con un significado distinto del propio pero que guarda cierta relación con el con el conjunto representado).

Al comenzar a ver la revista (a abrirla), y luego de una presentación que combina sonido y animación y actúa como preludio de los medios de expresión que explota la revista, el lector ve algo no demasiado diferente a lo que esperaría ver en el medio escrito: Una página principal dividida en zonas tituladas "Cine", "Libros", "Exposición", "Información sobre CLACSO", "Ayuda" y "Editorial" que son aproximadamente las mismas que tenía la revista original.

El porqué de una metáfora tan ortodoxa (la de una revista para representar una revista) para el punto de entrada es el de atenuar el choque ante la diferencia de medios. Ya habrá tiempo para una concepción radicalmente diferente.

El porqué del nombre Cine corresponde a la metáfora usada: Aquí se pretendía mostrar el potencial al máximo, y entonces era dónde mas crucial se hacía la facilidad y potencia en la representación. Se buscó que al mantener la coherencia en la metáfora, el lector pudiera realizar las acciones en un medio conocido (el cine) y entonces pudiera decidir por sí mismo en casos de indecisión.

La metáfora del cine actúa al traducirse la secuencia de presentación (la animación de un hombre acomodándose en su butaca, y un acercamiento a su mano con la que sostiene el programa de la función), la elección de la película en el programa y la visión de la película en una equivalencia la selección y lectura de los artículos.

Estos adoptan la forma de una película muda o fotonovela con el agregado de un "director" (el lector) que puede influir sobre su desarrollo.

La interface con la que el lector opera sobre los artículos está dividida en dos niveles: uno principal con las acciones básicas y uno secundario, que podríamos llamar el nivel Director.

La visión de los artículos se hace mediante el botón PLAY que muestra una "toma" (fondo y secuencia de "diálogos") sobre la que se ven paneles (algo así como los diálogos de una fotonovela o los cartones de las películas mudas) conteniendo texto, imágenes o sonido en forma secuencial y cantidad variable. Dado que los artículos incluídos en la revista estaban escritos para un medio común hubo que transformarlos: ahora los artículos están estratificados y tienen niveles de lectura a los que se asciende o desciende mediante links incluídos en los paneles. Cualquier nivel asegura el conocimiento de las ideas tratadas, pero distinto nivel de profundidad. Los links también proporcionan información accesoria (aclarando un párrafo, por ejemplo) o permiten conectar partes de los artículos que muestren alguna relación, quebrando la lectura secuencial.

A diferencia de las películas (aquí se nos puede acusar de violar la metáfora), el espectador puede desviar su atención en cualquier momento, adelantar o retroceder rápidamente en los paneles, sin obligación de leer previamente el panel anterior.

Si el usuario decide realizar un cambio, presiona "Corten!" que lo lleva a un panel con las opciones del modo Director. Puede ver rápidamente una "pelicula-artículo", puede finalizar la proyección abruptamente (volver a la imagen inicial de la página de la revista), puede comenzar la proyección de otra película (ver otro artículo), capturar una imagen (marcarla), ver una imagen capturada, retroceder

una "toma"(pantalla anterior), agregar un comentario o un link. Los comentarios permiten que el lector anote su opinión e ideas sobre los diálogos y las imágenes de la 'toma', mientras que los links permiten establecer relaciones arbitrarias que el lector considere importantes.

El resto de las secciones tiene las siguientes características:

LIBROS: Muestra un mesa con libros, permitiendo abrirlos, leer un resumen de su contenido y cerrarlos.

EXPOSICION: Contiene todas las imágenes de los artículos (índice de ilustraciones y datos de identificación).

AYUDA: Es un help on-line sensible al contexto.

Esperamos haber dado una idea de como se diferencia una reperesntación tradicional y una representación en hipermedia de la misma información y de como se amalgaman los nuevos elementos que los hipermedios permiten ofrecer a un lector que muta cada vez más a un papel de lector/autor activo.

BIBLIOGRAFIAS

Bolter, Jay. *Writing space.* Prentice Hall, 1990.
Miyazawa, Mitsumasa, et al. *An Electronic Book: APTBook.* Human Computer Interaction - Interact'90.
Simpson, Annette. *Lost in Hyperspace: how can designers help?.*Intelligent Tutoring Media, Vol. 1, Nro. 1, 1990.
Simpson, Annette. *The interface to a hypertex journal.* Human Computer Interaction - Interact'90.
Valdez, F.,M. Chignell,B. Glenn, *"Browsing Models for Hipermedia Databases"*, Proceedings of the Human Factory Society, 32nd Anual Meeting, 1988, pp. 318-322.

A Dense Instruction Set Computer Architecture Proposal

Olaf S. Schoepke
School of Mathematical Sciences
University of Bath
Bath, Avon BA2 7AY, United Kingdom

Abstract

In this paper we propose a dense instruction set architecture which offers some advantages over existing architectures. We approach the problem from different points of view: Entropy measurements show high redundancy in instruction streams, especially when treated as higher order Markov sources; the memory fetch time for instructions has been a major bottleneck for many years, especially for multiprocessors with shared memory systems; ever larger programs are in use which require more and more memory; networks often wasting their time transfering redundant information, and loading times, and bus traffic increase with larger programs. We suggest a computer architecture which can eliminate the problems mentioned above using dense instruction sets.

Keywords: Dense Instruction Set, Compact Code, Entropy

1 Introduction

This paper proposes a new computer architecture based on dense computer instruction sets built for RISC processors. Dense instruction set architectures are important primarily for three reasons: 1) Memory fetch time for instructions has been a bottleneck for many years, due to ever faster processors and multiprocessor systems with shared memory. 2) Entropy measurements show high redundancy in instruction streams, especially when treated as higher order Markov sources. 3) Program size is still increasing, resulting in longer loading times, more paging and swapping activities, and more memory–processor bus traffic. The memory–processor performance gap is clearly a problem on the horizon, and simply making caches larger cannot eliminate this problem. Therefore we introduce a dense instruction set computer architecture which offers some advantages over existing architectures due to much smaller program size. Arithmetic coding, a technique which can get as close as desired to the entropy, is used to encode and decode the instruction stream. Encoding has to be done on static code prior to execution, and decoding on dynamic code during execution.

2 Entropy of Instruction Sets

This section shows the program entropies gathered from observations of the SPARC[1] [10] architecture. Sample programs from which this information is collected are two C compilers, a debugger, part of a window system, part of the operating system and a statistics program. All these statistics are program dependent, because different classes of applications typically use different language features [11]. However, statistics about static characteristics of programs are based on more than 300,000 instructions, dynamic characteristics on millions of instructions.

Static characteristics of programs are interesting for code compaction, program loading times, swapping and paging, and transfer times over the network. Program size becomes more and more important since memory cycle time has not decreased with the same speed as the processor cycle time has increased, and the gap is still increasing. One of the first such studies of programming languages was published by Knuth in 1971 [4], but there are several other publications about static program behaviour.

Dynamic characteristics of programs [3, 8, 11] are of help in the area of program speed and the overall program run time. They are generally more interesting than static characteristics, but they are also more difficult to collect.

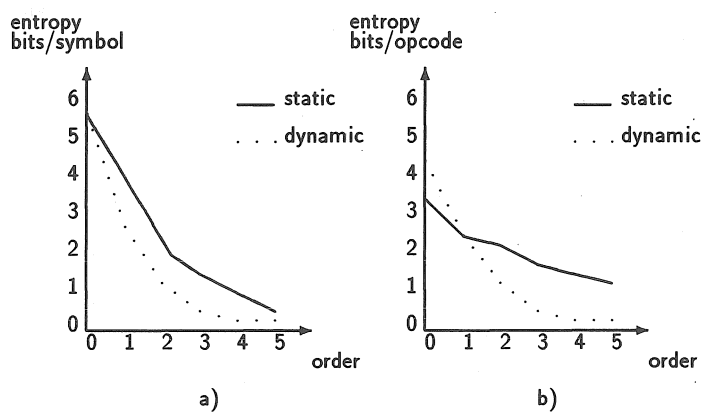


Figure 1: Average entropy of m-th order Markov source in bits/symbol and bits/opcode.

Figure 1 shows the calculated entropy for a m-th order Markov source [1] for the static as well as the dynamic case built with eight bit long symbols in case a) and opcodes [10] in case b) from the instruction stream. For eight bit long symbols as well as opcodes

[1]SPARC is a trademark of Sun Microsystems, Inc.

entropy decreases significantly for the fifth order Markov source from the zero order Markov source. These results show that there is great redundancy in such instruction streams and great dependency between instructions following each other which leads to very high conditional probabilities and thus to very low entropies. Such low entropies as shown in Figure 1 imply we could generate more compact code [9] because the entropy $H(S)$ gives the number of bits for optimal encoding. It is not surprising that we reach a lower entropy in the dynamic case than in the static case, because equal instruction sequences appear more often, e.g. in loops or frequently called procedures.

3 Arithmetic Coding

Recently, arithmetic coding [6, 13] has aroused increased interest as a technique that gives high compression efficiency for a variety of data types as well as being amenable to efficient software and hardware implementations [2]. Arithmetic coding is a coding technique which can get as close as desired to the entropy. It is easy to see that symbols with high probabilities of occurrence and hence larger intervals have less effect on narrowing the given interval than symbols with smaller probabilities.

Arithmetic coding is a data compression technique that encodes data by creating a code string which represents a fractional number between 0 and 1. As the message becomes longer, the interval needed to represent it becomes smaller, and the number of bits needed to specify that interval is growing. Successive symbols of the message reduce the size of the interval in accordance with the symbol probabilities generated by the given model. The more likely symbols reduce the range by less than the unlikely symbols and hence add fewer bits to the message.

Due to the complexity of most compression methods, past implementations of data compression techniques have mostly been restricted to software. A hardware design of the LZW algorithm using hash tables is briefly discussed in [12] and a fast VLSI implementation of the Huffman's scheme is given in [7]. The arithmetic coding scheme consists of arithmetic operations like addition and multiplication which can be implemented in hardware. Bassiouni [2] describes a high level VLSI chip for the implementation of arithmetic coding.

4 Suggested Algorithms for Encoding and Decoding

The encoding process must be preceded by post compiling to add information necessary to decode the given instruction stream. The algorithms give a brief overview on how the compiling has to be done to detect branch targets during the encoding process

```
Encoding_Program ()
{
  Initialize_Arithmetic_Encoder ();
  while !( EOF )
  {
     read ( instruction );
     if ( instruction == branch_target )
       Encoding_Procedure ( instruction );
     else
       Encode_Instruction ( instruction );
  }
}

Encoding_Procedure ( instruction )
{
  End_Current_Arithmetic_Encoding_Phase();
  Insert_Decode_Instruction ();
  Initialize_Arithmetic_Encoder ();
  Encode_Instruction ( instruction );
  return;
}
```

Figure 2: Encoding Algorithms

and how the decoder can examine the dense instruction stream to provide instructions to the execution unit for correct execution.

As we can see in the encoding algorithms provided in Figure 2 a new phase of encoding has to be started at each branch target. This of course influences the compression results especially as we know that only a few number of instructions are between branch instruction and branch target.

The decoding algorithms (Figure 3) show us a brief summary of the decoding and execution phase. Building an instruction depends on the number of instruction streams, the size of the symbol to be decoded and the type of the instructions to be built. For example building a 32 bit instruction needs 4 symbols each 8 bit long.

5 Instruction Execution

In this section we take a closer look at how instructions are executed. First we examine the common phases for von Neumann architectures to execute an instruction, and second, we show how to execute dense object code.

Common Instruction Execution

A program for a von Neumann machine consists of a set of instructions that are

```
Decoding_Program ()
{
  Initialize_Arithmetic_Decoder ();
  read ( symbol );
  while !( EOF )
  {
     Decode_Symbol ( symbol );
     Build_Instruction ( symbol );
     if ( instruction == branch_target )
       Initialize_Decoder ();
     else
       if ( instruction == branch_instruction )
         Decoding_Procedure ( instruction );
       else
         Execute_Instruction ( instruction );
    read ( symbol );
  }
}

Decoding_Procedure ( instruction )
{
  Execute_Branch_Instruction ();
  if !( branch_taken )
    return;
  else
  {
    Initialize_Arithmetic_Decoder ();
    Determine_New_Symbol_Address ();
    return;
  }
}
```

Figure 3: Decoding Algorithms

examined one after another; a program counter (PC) in the control unit indicates the next location in memory from which an instruction is to be taken. The vast majority of present–day computers are von Neumann machines. The three distinct phases that constitute the sequencing of each instruction for these architectures are: 1) Determining the memory address that contains the instruction. 2) Fetching the instruction from memory and 3) Executing the instruction.

Before a processor can execute an instruction it must fetch the instruction from memory. Before this operation, the instruction pointer has to be updated. If a branch has been taken, the address of the next instruction to be executed has to be calculated. Therefore, the rate at which the processor executes instructions cannot exceed the rate

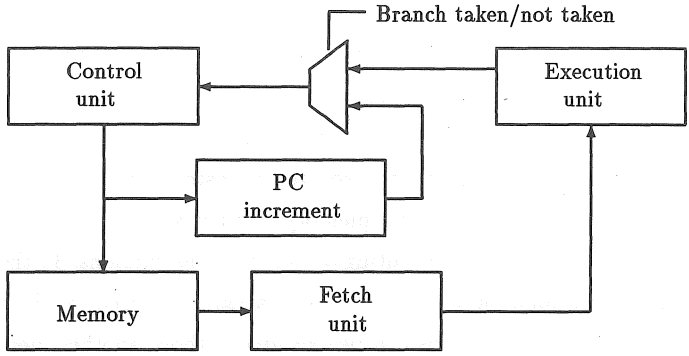


Figure 4: Instruction Execution

at which instructions are fetched from memory. During the execution phase of each instruction, the processor determines the memory location of the next instruction to be executed. Therefore, the memory does not operate independently of the processor. Figure 4 shows an example of a von Neumann architecture.

Executing Dense Instructions

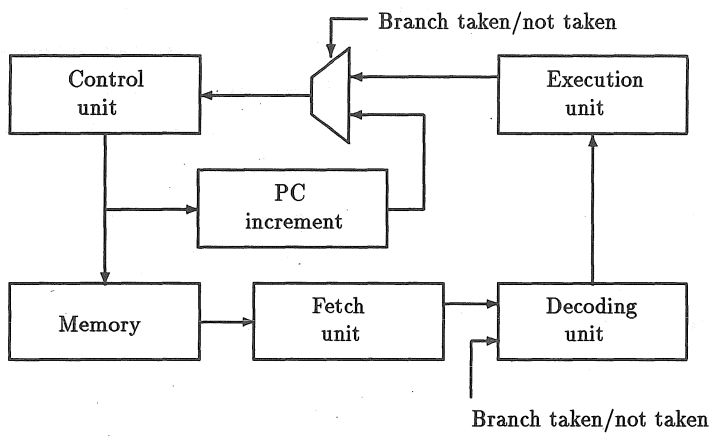


Figure 5: Executing Dense Instruction Sets

Executing dense instruction sets is more complicated than executing conventional object code. Operating independently of the execution unit the fetch unit fetches the instruction from memory. The decoding unit decodes the instruction and stores them for the execution unit to read. This could also be done in a pipeline structure to achieve

better performance.

Before executing an instruction new information has to be fetched if required from the main memory through the fetch unit into the decoder. In the decoding unit, the byte stream has to be decoded and transformed into an executable instruction. Having done that, the execution unit is able to execute the instruction. Because of the dense instruction stream provided by the main memory, it is not always necessary to fetch new information for the decoder after executing one or more instructions. In the worst case, e.g. after taking conditional branches, new information is required for the decoder, but in the best case, several instructions can be executed without another memory access.

Figure 5 gives us an example of a suggested architecture to execute dense object code with only one instruction stream to be decoded. For decoding several streams reflecting the structure given in an instruction stream, the given architecture has to be multiplied by the number of streams to be decoded including an on chip connection between the decoding units for communication. Fetch and decoding units can operate independently of the execution unit. Therefore, the decoding unit can decode the next instruction in advance like in any other pipeline structure.

One of the most difficult problems is how to handle branches, because branch instructions alter the control flow of the program from sequential execution. Like sequencing instructions [5] and cache memories, dense object code execution suffers after having taken branches, because the new memory address has to be calculated and new information is necessary to continue decoding the instruction stream. Furthermore we lose some structure given in the instruction stream because we cannot rely on information further back. The decoding algorithm has to be restarted with only little knowledge of the context and at an addressable memory location.

6 Conclusion

Because the memory–processor performance gap is clearly a problem in the near future, architectures are needed which can speed up the supply of instructions to the execution. We suggest a new architecture based on dense instruction sets which offers some advantages over existing architectures. On the one hand, the instruction stream has to be decoded before the instruction can be executed by the execution unit. On the other hand, the time used to transfer data especially between several processors and shared memory will drop sharply. Smaller program size also means less swapping, shorter loading times, and minor traffic on a network. With the method presented, it is possible to reduce program size considerably and thus to narrow the increasing gap

between memory cycle time and processor cycle time.

References

[1] N Abramson *Information Theory and Coding* McGraw–Hill, 1963

[2] MA Bassiouni, N Ranganathan, A Mukherjee *Software and Hardware Enhancement of Arithmetic Coding* Lecture Notes in Computer Science, Fourth International Working Conference SSDBM, Rome, Italy, June 1988

[3] JL Hennessy, DA Patterson *Computer Architecture A Quantitative Approach* Morgan Kaufmann Publishers Inc., 1990

[4] DE Knuth *An Empirical Study of FORTRAN Programs* Software Practice and Experience, pp. 105–133, 1971

[5] RF Krick, A Dollas *The Evolution of Instruction Sequencing* IEEE Computer, pp. 5–15, April 1991

[6] GG Langdon *An Intr. to Arithmetic Coding* IBM J. Res. Dev., vol 28 no 2, 1984

[7] A Mukherjee, M Bassiouni *A VLSI chip for efficient transmission and retrieval of information* Proc. 10th ACM SIGIR International Conf. on Research and Development in Information Retrieval, pp. 208–216, June 1987

[8] DA Patterson, CH Sequin *A VLSI RISC* Computer, pp. 8–21, September 1982

[9] OS Schoepke *Using the Entropy in the SPARC Instruction Set* IEEE Proc. of the International Conference on Computing and Information, May 1992

[10] Sun SPARC Architecture Manual Revision A, October 1987

[11] RP Weiker *Dhrystone: A Synthetic Systems Programming Benchmark* Communications of the ACM, vol 27 no 10, October 1984

[12] TA Welch *A Technique for High–Performance Data Compression* Computer, pp. 8–19, June 1984

[13] IH Witten, RM Neal, JG Cleary *Arithmetic Coding for Data Compression* Communications of the ACM, vol 30 no 6, June 1987

APLICACIONES MATEMATICAS EN EL DISEÑO DE SISTEMAS TELEINFORMATICOS

Mat. Teresa Silva Ruales
Centro de Matemáticas
Universidad Central del Ecuador

Ing. Marleni Pineda Q.
Escuela de Ciencias
Facultad de Ingeniería
Universidad Central del Ecuador

El presente trabajo propone una metodología de diseño y dimensionamiento de un sistema teleinformático, basado en el estudio de sus componentes y apoyado en una base matemática. Esta metodología incluye tópicos referentes al estudio probabilístico de las transacciones para determinar el volumen de información a ser manejado, al dimensionamiento de una red de teleproceso en lo que se refiere al número de terminales, líneas de comunicación y las características del sistema central.

1. PRESENTACION

El Diseño de Sistemas Teleinformáticos en Latinoamérica es realizado exclusivamente por empresas que comercializan equipos informáticos, quienes en vez de realizar un diseño en función de los requerimientos reales del usuario lo sobredimensionan y equivocan el equipo sugerido, lo cual constituye una desventaja puesto que implica un desperdicio de recursos. El usuario de un Sistema Teleinformático puede conocer los requerimientos del mismo en base a un estudio de sus componentes y a una evaluación de los equipos ofrecidos.

Nuestro objetivo es presentar una alternativa para una adecuada elección del equipo informático, para lo cual hemos seguido una Metodología de Diseño de Sistemas Teleinformáticos [1], la misma que está apoyada en una base matemática, identificando los modelos matemáticos y los parámetros utilizados para construir un algoritmo que permita la mecanización del diseño de una red de teleproceso, resultando un programa de fácil manejo sin que se requiera tener conocimientos especializados sobre matemática e informática.

2. MODELOS MATEMATICOS EMPLEADOS

La base matemática para el Diseño de Sistemas Teleinformáticos se refiere fundamentalmente a la Teoría de Probabilidades y Teoría de Colas, las cuales se emplean en la evaluación de los diferentes componentes del sistema gracias a la naturaleza aleatoria de las variables del mismo y de las que intervienen en cada relación dentro del sistema. En general, para aplicar la teoría de colas, en el desarrollo analítico se considera el sistema en estado estacionario

A continuación presentamos en forma resumida las principales ecuaciones matemáticas que se utilizan en el desarrollo de esta metodología, en lo que se refiere a la Teoría de Colas.

Los parámetros de un sistema de colas se especifican según la siguiente nomenclatura: **A/B/m/k/l**, donde A - función de distribución de tiempos entre llegadas. B - función de distribución de los tiempos de servicio. m - número de servidores independientes. k - capacidad de almacenamiento de la cola, si falta se supone infinita. l - población de usuarios, si falta se supone infinita. Los indicadores A y B suelen ser la inicial de la distribución en cuestión, así por ejemplo las distribuciones más usadas en teoría de colas son: distribución exponencial (M), determinista (D), Erlang de k etapas (E_k) y cualquier distribución (G).

- Resultado de Little: "El número medio de usuarios en un sistema de colas es igual a la tasa de llegadas de usuarios al sistema por el tiempo medio pasado en ese sistema". $\overline{N}_T = \lambda T_T$

- Intensidad de tráfico: Cociente entre el tiempo medio de servicio y el tiempo medio entre llegadas. $IT = \frac{T_s}{T_a} = \lambda T_s = \frac{\lambda}{\mu}$

- Factor de utilización: Fracción media de servidores ocupados. $\rho = \min\{\frac{\lambda}{\mu}, 1\}$. Si hay un solo servidor: IT=ρ, para m servidores: IT=mρ.

- Tiempo medio de respuesta: Tiempo que un usuario pasa en el sistema, esto es la suma del tiempo medio de espera en la cola y el tiempo medio de servicio. $T_T = T_e + T_s$

- Longitud de la cola: Número de usuarios en la cola en el instante t. $\overline{N}_e = \lambda T_e$

- <u>Cola M/M/1</u>:

  Probabilidad de que se encuentren N usuarios en el sistema: $P_N = (1-\rho)\rho^N$

  Número medio de usuarios en la cola: $\overline{N}_e = \frac{\rho^2}{1-\rho}$

  Tiempo medio de espera en la cola: $T_e = \frac{\rho T_s}{1-\rho}$

  Número medio de usuarios en el sistema: $\overline{N}_T = \frac{\rho}{1-\rho}$

  Tiempo medio de respuesta: $T_T = \frac{T_s}{1-\rho}$

- Cola M/G/1:

Siendo m_2 y σ_s^2, el momento de segundo orden y la varianza del tiempo de servicio respectivamente, se tiene que:

$$\bar{N}_c = \frac{\lambda^2 m_2}{2(1-\lambda)} + \rho \qquad T_c = \frac{\lambda m_2}{2(1-\rho)} \qquad T_T = \frac{T_s}{1-\rho}\left[1-\frac{\rho}{2}\left(1-\frac{\sigma_s^2}{T_s^2}\right)\right]$$

- Cola M/M/m:

Probabilidad de que estén k usuarios en el sistema:

$$P_k = \begin{cases} \dfrac{(\lambda T_s)^k}{k!} P_0, & k \le m \\ \dfrac{(\lambda T_s)^k}{m!\, m^{k-m}} P_0, & k > m \end{cases}$$

Probabilidad de que el sistema esté vacío:

$$P_0 = \left[\sum_{k=0}^{m-1} \frac{(\lambda T_s)^k}{k!} + \frac{(\lambda T_s)^m}{m!}\frac{m}{m-\lambda T_s}\right]^{-1}$$

Utilización del sistema: $\rho_s = \dfrac{\rho}{m}$

Número medio de usuarios en el sistema: $\bar{N}_T = \lambda T_s = IT$

Probabilidad de que un usuario deba esperar: $P_c = \dfrac{\rho^m/m!}{\dfrac{\rho^m}{m!} + (1-\rho_s)\displaystyle\sum_{k=0}^{m-1}\frac{\rho^k}{k!}}$

Número medio de usuarios en la cola: $\bar{N}_c = \dfrac{\rho_s}{1-\rho_s} P_c$

Tiempo medio de espera en la cola:

$$T_c = \frac{P_c T_s}{m(1-\rho_s)} = \frac{P_c}{\rho} T_{c_1} = \frac{P_c}{\rho}\frac{\rho_s}{1-\rho_s} T_s$$

Modelo del Reparador

Relación de servicio: $Z = \dfrac{T_f}{T_s}$, donde T_f es el tiempo de funcionamiento y T_s el tiempo de servicio del reparador.

Probabilidad de que k máquinas estén averiadas: $P_k = \dfrac{\dfrac{e^{-z} z^{m-k}}{(m-k)!}}{e^{-z}\displaystyle\sum_{j=0}^{m}\frac{z^j}{j!}}$

Utilización del reparador: $\rho_R = 1 - \frac{z^m/m!}{\sum_{j=0}^{m} \frac{z^j}{j!}}$

Número de máquinas en la cola de reparación: $\bar{N}_e = m - (Z+1)\rho_R$

Número medio de máquinas averiadas: $N_T = m - z\rho_R$

Tiempo medio entre fallos: $T_F = T_e + T_s + T_f = \frac{mT_s}{\rho_R}$

Tiempo medio que una máquina está averiada: $T_T = T_e + T_s = \frac{mT_s}{\rho_R} - T_f$

$$\frac{T_F}{T_f} = \frac{m\sum_{j=0}^{m} \frac{z^j}{j!}}{\sum_{j=1}^{m-1} \frac{z^{j+1}}{j!}} = \frac{m}{\rho_R Z}$$

2. MECANIZACION DE LA METODOLOGIA DE DISEÑO DE SISTEMAS TELEINFORMATICOS

A continuación se presenta un esquema general para la mecanización de la metodología antes mencionada mediante actigramas, los cuales se encuentran etiquetados en forma jerárquica. Un actigrama tiene la estructura de la figura 1.

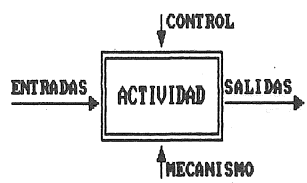
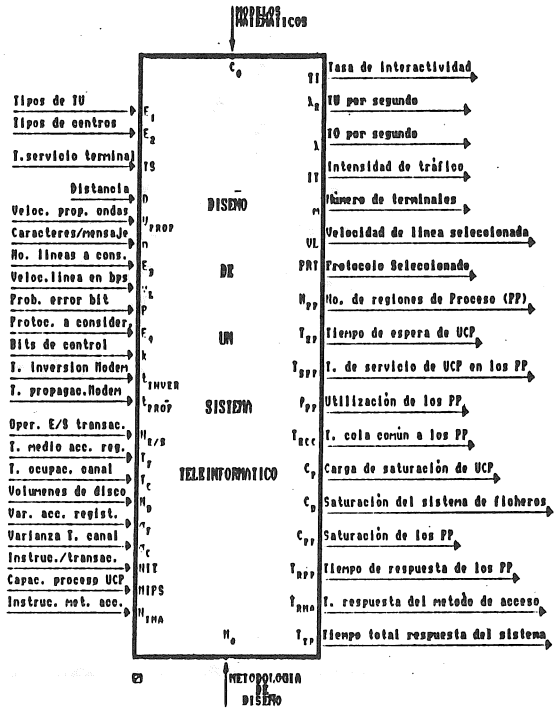


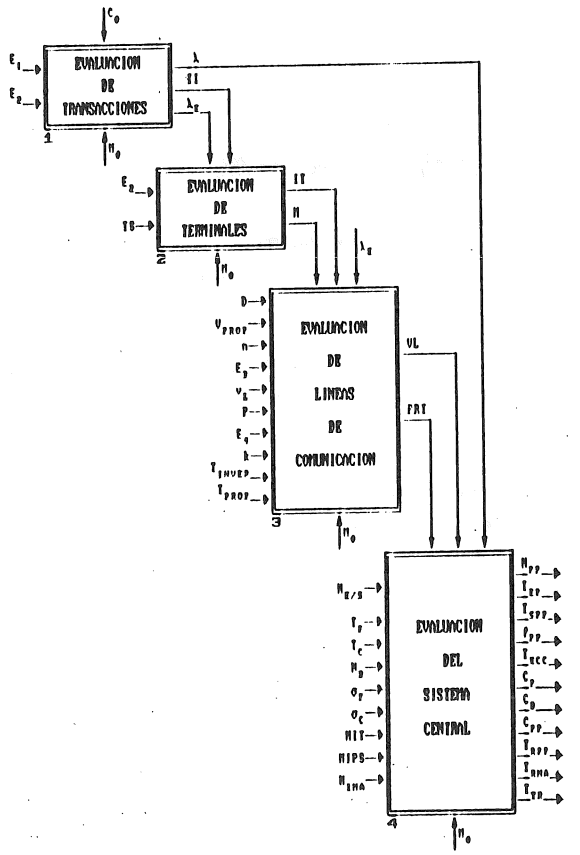
Fig. 1. Estructura de un actigrama

En el actigrama general A, se puede apreciar la función principal de esta aplicación. El actigrama A0 nos indica las fases de tratamiento, que son las siguientes:

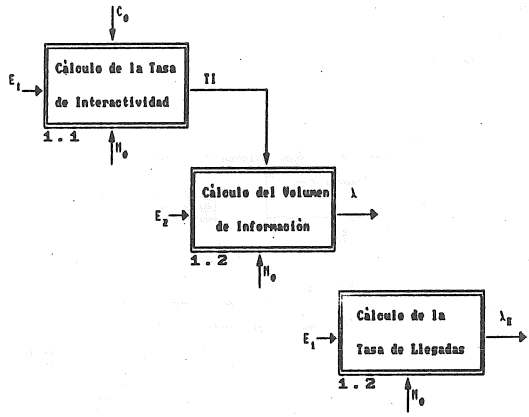
A1: Evaluación de transacciones
A2: Evaluación del número de terminales
A3: Evaluación de las líneas de comunicación
A4: Evaluación del Sistema Central



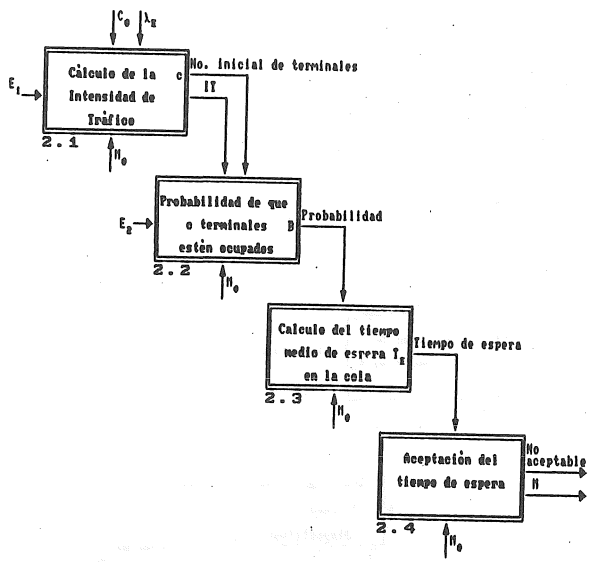
A: Programa de Aplicación para la Mecanización de la Metodología de Diseño de Sistemas Teleinformàticos



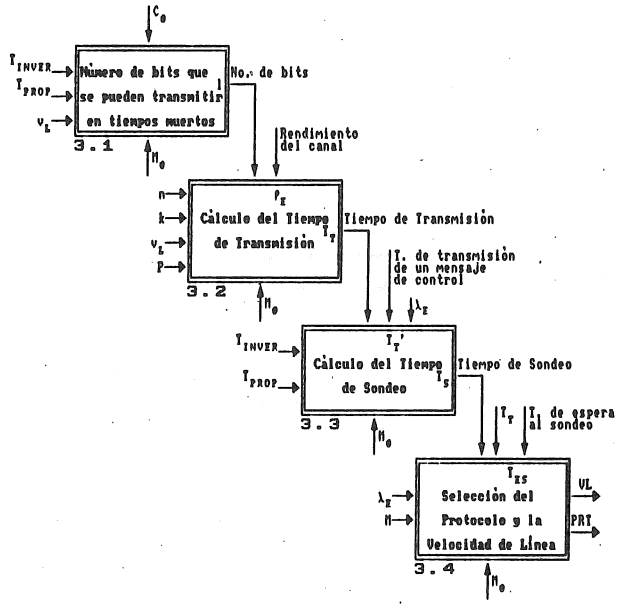
AA: Diseño de un Sistema Teleinformàtico



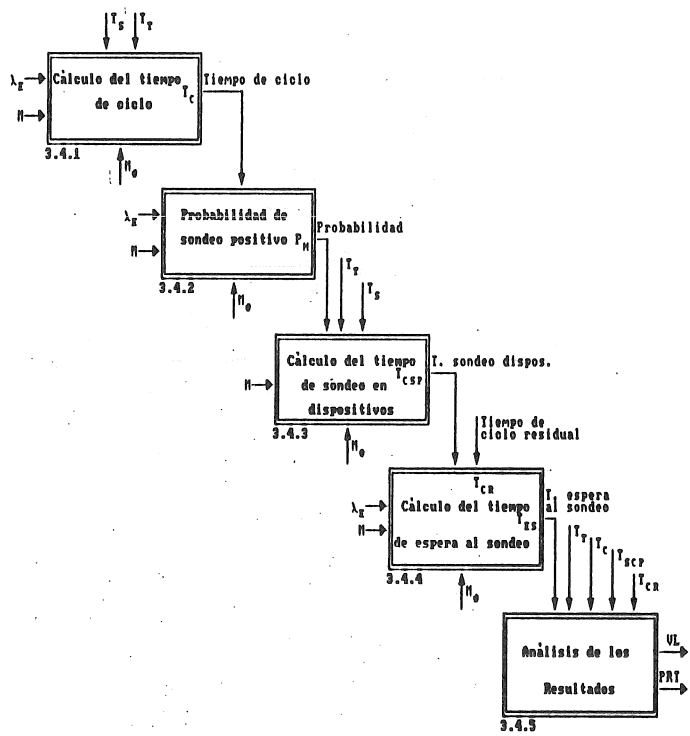
A1: Evaluación de Transacciones



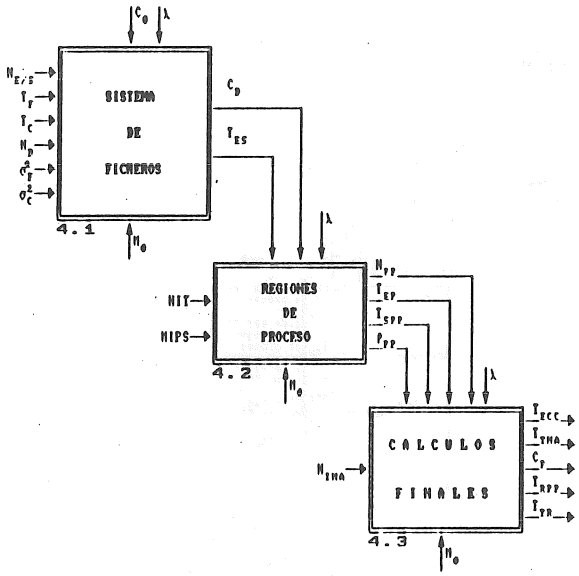
A2: Evaluación del Número de Terminales



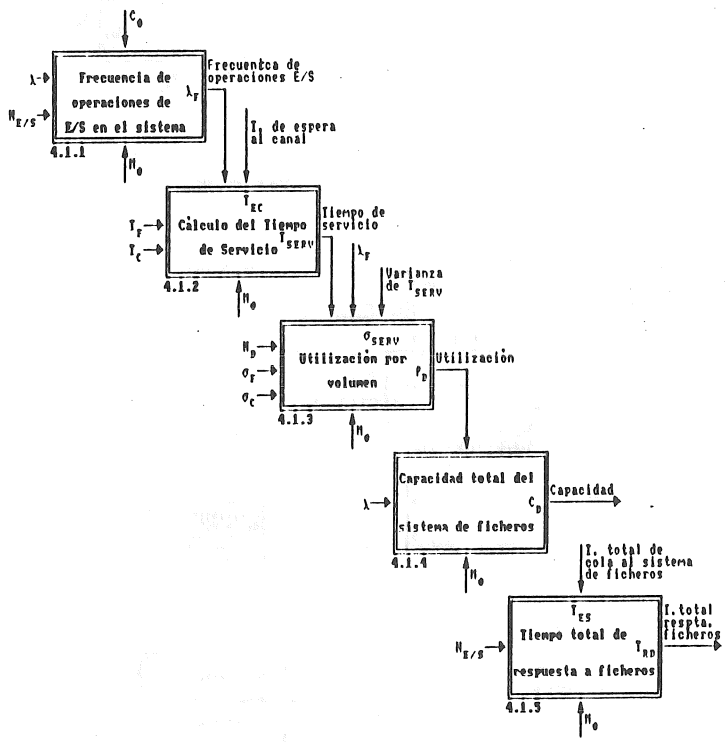
A3: Evaluación de las Líneas de Comunicación



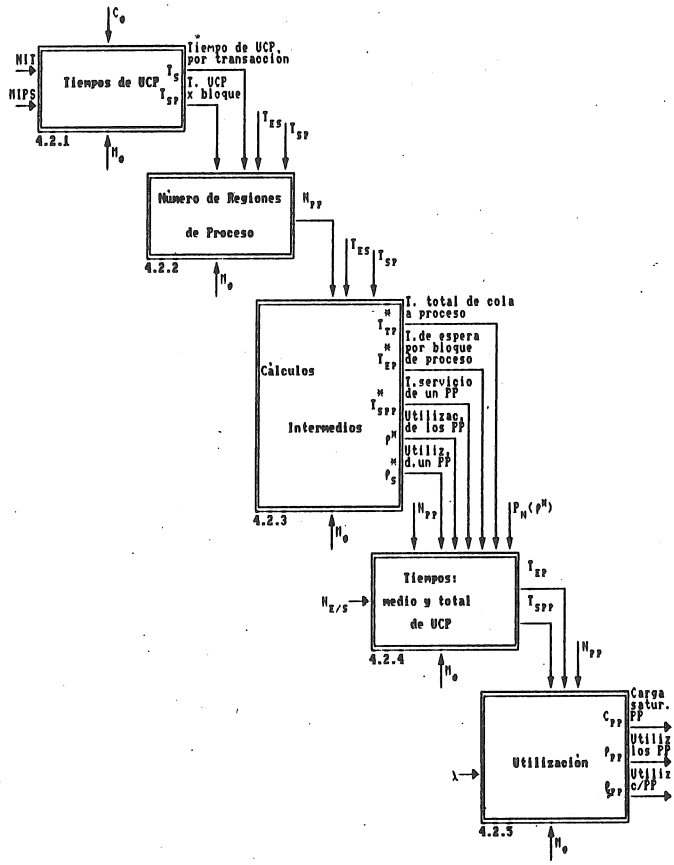
A34: Selección del protocolo y velocidad de línea



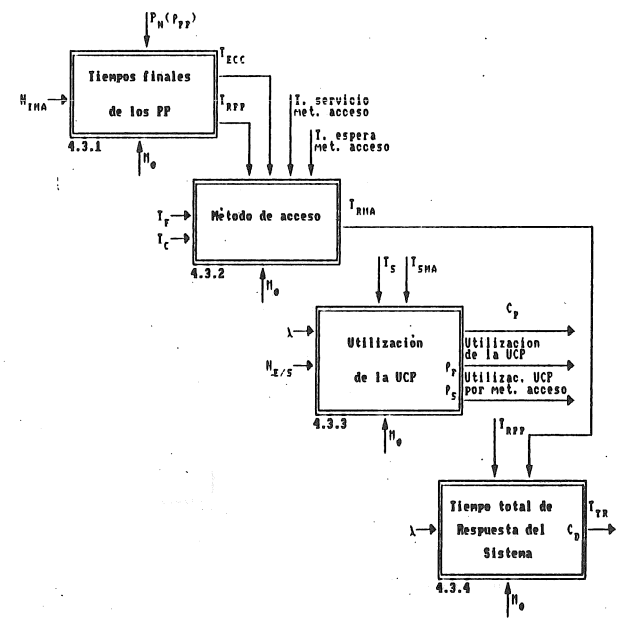
A4: Evaluación del Sistema Central



A41: Sistema de Ficheros



A42: Regiones de Proceso



A43: Calculos Finales

A1: Evaluación de Transacciones

Función: Estudio probabilístico de las transacciones en base a datos obtenidos de un análisis funcional de los requerimientos del sistema se determina el volumen de información a ser manejado (transacciones de operador/segundo), la tasa de interactividad (transacciones de operador/transacción de usuario). Además encuentra el número de transacciones de operador y la tasa de interactividad de cada centro y del sistema en general.

Datos:

- Número de tipos de transacciones de usuario, TU, (NTU).
- Número de tipos de centros (NC).
- Nombre de cada tipo de TU.
- Número de transacciones de operador (TO) por cada TU (TO_j).
- Nombre de cada tipo de centro y número de agencias.
- Número de TU por cada centro (TU_{ji}, transacciones tipo j para el centro i).

Operaciones:

- Cálculo de la tasa de interactividad o número de transacciones de operador para cada tipo de centro.

$$TI_i = \frac{No.TO}{No.TU} = \frac{\lambda(TO)_i}{\lambda(TU)_i}$$

- Sea TUi el número total de transacciones en el centro i, Gi, el número de agencias en el centro i; entonces:

$$\lambda(TU)_i = \sum_{j=1}^{NTU} TU_{ji}$$

$$\lambda(TO)_i = \sum_{j=1}^{NTU} TO_j TU_{ji}$$

- Cálculo de las transacciones de operador por cada tipo de centro.
  $TOC_i = \lambda(TU)_i * TI_i * G_i$
- Cálculo de las transacciones de operador del sistema, resultando del número de transacciones diarias.

$$TO_{SISTEMA} = \sum_{i=1}^{NC} TOC_i$$

- Cálculo del número de transacciones de operador y de usuario ($TO_{SISTEMA}$) por hora, o y segundo.
  λ = Número de transacciones por segundo
  λ_E = Número de transacciones de usuario por segundo

A2: Evaluación del Número de Terminales

Función: Consiste en dimensionar la red de teleproceso en lo que se refiere al número de terminales de acuerdo al volumen de información calculado y al tiempo medio de espera en la cola, el mismo que se calcula mediante un

proceso iterativo utilizando las ecuaciones de una cola con servidores múltiples.

Datos:
- Tiempo de servicio de un terminal (T_s), en cada centro.

Operaciones:
- Cálculo de la intensidad de tráfico: $\rho = \lambda_E T_s$
- Número inicial de terminales: $c = \rho$
- Cálculo iterativo para encontrar el número de terminales por cada centro

$$c = \frac{BT_s}{T_e(1-\rho_T)}$$

  . Cálculo del porcentaje de ocupación del terminal: $\rho_T = \frac{\lambda_E}{c} T_s$

  . Búsqueda de la probabilidad de que los c terminales estén ocupados, aplicando la ecuación de los sistemas con servidores múltiples:

$$B = \frac{\frac{\rho^c}{c!}}{\frac{\rho^c}{c!} + (1-\rho_T)\sum_{i=0}^{c-1}\frac{\rho^i}{i!}}$$

  . Búsqueda del tiempo medio de espera en la cola: $T_e = \frac{BT_s}{c(1-\rho_T)}$

  . Incremento del número de terminales para seguir con los cálculos hasta que T_e sea aceptable.
- Una vez que T_e es aceptable, c es el número de terminales.

A3: Evaluación de Líneas de Comunicación

Función: Tiene como objetivo encontrar el tipo de línea de comunicación y el protocolo de línea que debe usar el sistema teleinformático, evaluando las características de los tipos de líneas ofertadas para que el usuario pueda elegir la línea y protocolo más convenientes a sus necesidades.

Datos:
- Distancia a considerar (distancia)
- Velocidad de propagación de las ondas según el tipo de línea (V_{prop})
- Número promedio de caracteres por mensaje (n)
- Datos de las líneas de comunicación a evaluarse:
  . Número de líneas que se considerarán
  . Velocidad de cada línea en bps (v_i)
- Probabilidad de error por bit (10^{-5} para líneas dedicadas y 10^{-4} para líneas conmutadas).

- Datos de los protocolos de línea:
  - . Número de tipos de protocolos que se considerarán
  - . Nombre de cada protocolo
  - . Número de bits de control (k)
- Datos del modem:
  - . Tiempo de inversión en segundos (T_{inver})
  - . Tiempo de propagación en segundos (T_{prop})

Operaciones:

- Tiempo de propagación en la línea: $T_p = \frac{distancia}{V_{prop}}$
- Cálculo del número de bits que se pueden transmitir durante los tiempos muertos de inversión y propagación: $l = v_l(T_{inver} + T_p)$
- Cálculo del rendimiento del canal: $\rho_E = \frac{\frac{n}{v_l}}{\frac{(n+k+l)}{v_l} e^{(n+k)p}} = \frac{n}{n+k+l} e^{-(n+k)p}$
- Cálculo del tiempo de transmisión: $T_T = \frac{n+k+l}{v_l} e^{(n+k)p}$
- Cálculo del tiempo de transmisión para un mensaje de control: $T_t' = (k+l) \frac{e^{kp}}{v_l}$
- Cálculo del tiempo de sondeo: $T_s = 2(T_t' + T_{prop}) + T_{inver}$
- Selección del protocolo y la velocidad de la línea:
  - . Cálculo del tiempo de ciclo: $T_c = \frac{MT_s}{1-\lambda_E T_t}$
  - . Búsqueda de la probabilidad de sondeo positivo: $P_m = \frac{\lambda_E T_c}{M}$
  - . Cálculo del tiempo de sondeo en dispositivos: $T_{csp} = MT_s + T_t + P_m T_t(M-1)$
  - . Cálculo de la varianza de T_{csp}: $V_c = MP_m|E(t_t^2) - P_m T_t^2|$; $E(t_t^2) \approx T_t^2$
  - . Cálculo del tiempo de ciclo residual: $T_{cr} = \frac{E(t_c^2)}{2T_c} = \frac{T_c^2+V_c}{2T_c}$ (M/G/1)
  - . Cálculo del tiempo de espera al sondeo: $T_{es} = \frac{T_{cr}}{1-\lambda_E T_{csp}/M}$
- Impresión de los resultados encontrados, con opción a seguir calculando para otro protocolo y otra velocidad de línea.
- De acuerdo a los resultados, el usuario seleccionará la línea y el protocolo que más le convenga.

A4: Evaluación del Sistema Central

Función: Conociendo las características del equipo y el volumen de información a ser manejado se obtiene los tiempos de respuesta del sistema en base a una descomposición de las funciones de la unidad central en: sistema de ficheros, regiones de proceso y tiempos finales, adaptando las relaciones entre sus elementos a modelos de teoría de colas.

Datos:

- Sistema de ficheros
  - Número de transacciones por segundo (λ)
  - Número de operaciones de entrada/salida por transacción ($N_{E/S}$)
  - Tiempo medio de acceso a un registro de fichero por operación de E/S (T_f)
  - Tiempo medio de ocupación del canal por operación de E/S (T_c)
  - Número de volúmenes de disco (N_D)
  - Varianza de acceso a registro de ficheros (σ_f)
  - Varianza de tiempo de canal (σ_c^2)
- Regiones de Proceso
  - Número de instrucciones por transacción (NIT)
  - Capacidad de proceso de la UCP en millones de instrucciones por segundo MIPS)
  - Número de instrucciones del método de acceso por transacción (N_{IMA})

Operaciones:

SISTEMA DE FICHEROS:

- Frecuencia de operaciones de E/S en el sistema: $\lambda_F = \lambda N_{E/S}$
- Cálculo de Z (modelo del reparador): $r(Z) = \lambda_F T_c$ utilización del canal

$$r(i) = 1 - \frac{\frac{i^{ND}}{(ND)!}}{\sum_{j=0}^{ND} \frac{i^j}{j!}}$$

- Cálculo del tiempo de espera al canal: $T_{ec} + T_c = \frac{N_D}{\lambda_F} - ZT_c$
- Tiempo de servicio: $T_{SERV} = T_{ec} + T_c + T_f$
- Varianza T_{ec}: $\sigma_{ec}^2 = T_{ec}^2$
- Varianza T_{SERV}: $\sigma_{SERV}^2 = \sigma_f^2 + \sigma_{ec}^2 + \sigma_c^2$
- Utilización por volumen: $\rho_D = \frac{\lambda_F}{N_D} T_{SERV}$

- Capacidad total del sistema de ficheros: $C_D = \frac{\lambda}{\rho_D}$

- Tiempo total de cola al sistema de ficheros:

$$T_{es} = \frac{T_{SERV}}{1-\rho_D}\left[1-\frac{\rho_D}{2}\left(1-\frac{\sigma^2_{SERV}}{T^2_{SERV}}\right)\right]$$

- Tiempo total de respuesta a ficheros: $T_{RD} = N_{E/S}T_{es}$

REGIONES DE PROCESO (PP):

- Tiempo de UCP por transacción: $T_s = \frac{NIT}{MIPS}10^{-6}$

- Tiempo de UCP por bloque: $T_{sp} = \frac{T_s}{N_{E/S}+1}$

- Número de regiones de proceso: $N_{pp} \approx \frac{T_{ES}}{T_{sp}}$

- Tiempo total de cola a proceso: $r_{N_{pp}}(Z) = \frac{\frac{Z^{Npp}}{N_{pp}!}}{\sum_{j=0}^{N_{pp}}\frac{Z^i}{i!}}, \quad Z = \frac{T_{es}}{T_{sp}}$

- Tiempo de espera por bloque de proceso: $T^*_{Tp} = T^*_{Ep}+T_{sp} = \frac{N_{pp}T_{sp}}{r_{N_{pp}}(Z)} - T_{ES}$

siendo $Z = \frac{T_{ES}}{T_{sp}}$ y $r_{N_{pp}}(Z)$ se calcula. $T^*_{EP} = T^*_{TP}-T_{sp}$

- Tiempo de servicio de un PP: $T^*_{spp} = T_{ES}N_{E/S}+(T^*_{Ep}+T_{sp})(N_{E/S}+1)$
- Utilización de las PP: $\rho^* = \lambda T^*_{spp}$
- Utilización de un PP: $\rho^*_s = \frac{\rho^*}{N_{pp}}$
- Probabilidad de que los PP estén ocupados:

$$P_N(\rho^*) = \frac{\frac{\rho^{*N_{pp}}}{N_{pp}!}}{\frac{\rho^{*N_{pp}}}{N_{pp}!}+[1-\rho^*_s]\sum_{k=0}^{N_{pp}-1}\frac{\rho^{*k}}{k!}}$$

- Tiempo medio de espera por bloque de UCP:

$$T_{Ep} \approx P_N(\rho^*)\,T^*_{Ep}+[1-P_N(\rho^*)]\,\frac{T^*_{Ep}}{2}$$

- Tiempo de servicio total de UCP en los PP:

$$T_{spp} = T_{RD} + (T_{Ep} + T_{sp})(N_{E/S} + 1)$$

- Carga de saturación de los PP: $C_{pp} = \frac{N_{pp}}{T_{spp}}$
- Utilización de la regiones de proceso: $\rho_{pp} = \lambda T_{spp}$
- Utilización de cada PP: $\rho_{spp} = \frac{\rho_{pp}}{N_{pp}}$

CALCULOS FINALES

- Tiempo de respuesta de los PP: $T_{RPP} = T_{sPP} + T_{ecc}$
- Tiempo de espera en la cola común a los PP

$$P_N(\rho_{pp}) = \frac{1 - r_N(\rho_{pp})}{1 - \rho_{spp} r_N(\rho_{pp})}$$

$$T_{Ecc} = \frac{P_N(\rho_{pp})}{\rho_{pp}} T_{Ecc1} = \frac{P_N(\rho_{pp})}{\rho_{pp}} \frac{\rho_{spp}}{1 - \rho_{spp}} T_{spi}$$

- Tiempo total del método de acceso
  .Tiempo de servicio del método de acceso: T_{SMA} = NIMA/MIPS
  .Tiempo de espera al método de acceso: $T_{EMA} = \frac{\rho_s}{(1-\rho_s)} T_{SMA}$

  $T_{RMA} = T_{EMA} + T_{SMA}$
- Carga de saturación del sistema: $C_p = \frac{1}{T_s + T_{SMA}}$
- Utilización de la UCP: $\rho_p = \frac{\lambda}{C_p}$
- Utilización de la UCP por método de acceso: $\rho_s = T_{SMA} N_{E/S}$
- Tiempo total de respuesta del sistema: $T_{TR} = (T_{Ecc} + T_{sPP}) + (T_{EMA} + T_{SMA})$.

BIBLIOGRAFIA

[1] GARCIA, J. y GONZALEZ, J., Diseño de Sistemas Teleinformáticos. Madrid. 1982

[2] HOPPER, A., TEMPLE, S. y WILLIAMSOM, R. Diseño de Redes Locales, Traducido por Héctor Carranco Ortiz, Addison Wesley. 1986.

[3] KLEINROCK, L., Queueing Systems Vol I y II. Wiley and Sons. 1976

[4] KOBAYASHI, H., Modeling and Analysis, Wesley. 1978.

[5] MORGAN, B. J. T., Elements of Simulation, Chapman and Hall. 1984.

[6] SAATY, T. L., Elementos de la Teoría de Colas, versión española de Rafael Pro. Madrid Editorial Juan Bravo. 1967.

QUICK-DB: Um Ambiente para Projeto e Manipulação de Bases de Dados

PEDRO MANOEL SILVEIRA

Núcleo de Computação Eletrônica — UFRJ
Caixa Postal 2324, Cep 20001, Rio de Janeiro, RJ, Brasil
E-mail: pedro@ufrj.bitnet

RESUMO

Este trabalho apresenta QUICK-DB, um ambiente para projeto, manutenção e manipulação de bancos de dados construídos sobre uma extensão do modelo Entidade/Relacionamento. QUICK-DB é composto por um conjunto de ferramentas de software que o tornam prático, simples e didático no desenvolvimento de aplicações de bancos de dados. Estas ferramentas incluem diagramador gráfico, dicionário de dados, folheador, interface para regras de produção, gerador de aplicações, servidor de bancos de dados e gerador de esquemas.

ABSTRACT

This article presents QUICK-DB, a graphical environment for project, maintenance and manipulation of databases built over an extension of the Entity/Relationship Model. A set of software tools make QUICK-DB simple, practical and comprehensive for the development of database aplications. These tools include the schema drawer, data dictionary, browser, expert systems shell, program generator, database server and database schema generator.

1. INTRODUÇÃO.

Este trabalho apresenta os fundamentos de QUICK-DB, um ambiente completo para projeto, manutenção e manipulação de bancos de dados construídos sobre o Modelo Entidade/Relacionamento. QUICK-DB é composto por um conjunto de ferramentas de software que o tornam prático, simples e didático no desenvolvimento de aplicações de bancos de dados.

QUICK-DB está baseado na idéia de um ambiente uniforme e unificado, gráfico, onde todos os componentes referem-se a uma extensão do Modelo E-R. Este enfoque permite que seus usuários sigam uma linha de trabalho consistente, simplificando a metodologia e evitando a introdução de conceitos estranhos ao modelo básico.

2. FACILIDADES.

Pela utilização de QUICK-DB é possível:

- Definir diagramas do tipo Entidade-Relacionamento completos, incluindo extensões como atributos referenciais, atributos virtuais, acesso ordenado, generalização;
- Verificar diagramas;
- Construir automaticamente um dicionário de dados durante a especificação do diagrama;
- Gerar descrições de bancos de dados equivalentes aos diagramas para implementação em sistemas de grande porte;
- Gerar bibliotecas de objetos e métodos para acesso aos bancos de dados descritos nos diagramas;

- Definir sistemas de regras de produção para a construção de sistemas especialistas, fortemente acopladas ao bancos de dados;
- Definir atributos virtuais através de sistemas de cláusulas simples;
- Folhear os bancos de dados por sobre uma interface gráfica;
- Gerar aplicações simples para manutenção dos bancos de dados.
- ; Ter acesso remoto a outras bases.

Além das funções listadas acima, já implementadas, QUICK-DB prevê também a possibilidade de:

- Gerar relatórios automaticamente a partir das bases.

3. FUNCIONALIDADE.

A Figura 1 apresenta o diagrama funcional dos módulos de QUICK-DB. Todos os componentes interagem sobre interfaces similares, gráficas.

A utilização do Modelo E-R para definição e utilização de bases de dados está calcada sobre um conjunto de ferramentas de software, descritas resumidamente a seguir.

Diagramador E-R: O diagramador é uma ferramenta gráfica para especificação de diagramas E-R estendidos. O sistema trabalha com *mouse* e menus gráficos e direciona o usuário nas suas tarefas. É aqui que se configura a "face" gráfica de um sistema. As demais ferramentas tomam o diagrama aqui definido como base, estabelecendo a unidade visual que o sistema oferece.

Dicionário de Dados: O dicionário de dados é automaticamente construído por ocasião da especificação do diagrama do banco de dados. Orientado para o Modelo E-R, tem como caracterítica principal sua plena capacidade de representar elementos do Modelo E-R e extensões, tais como as abstrações de generalização e agregação. O acesso ao dicionário de dados é feito através de um conjunto de métodos e atributos.

Gerador de esquemas de bancos de dados: A partir do diagrama E-R, expresso pelo dicionário de dados, são geradas descrições da base de dados para implementação nas instalações correntemente em uso.[12] Da análise das construções definidas pelo usuário no diagrama E-R, este módulo gera a descrição mais adequada para implementação no SGBD objeto.

Servidor E-R: Todo acesso à base de dados é feito através do Servidor E-R. Este é constituído por um conjunto de objetos e métodos cuja manipulação permite acesso e operações sobre a base. Os programas de aplicação têm acesso ao banco de dados pela chamada às bibliotecas de funções que operam sobre o banco de dados através do servidor.

Banco de Dados Autônomo: O sistema é capaz de atuar como banco de dados autônomo, na própia estação de trabalho. Para tanto, há uma versão do servidor de banco de dados que manipula um sistema de arquivos local.

Tanto para o servidor como para o banco autônomo, os objetos , atributos e métodos são gerados de tal forma que os usuários podem ter acesso ao banco e ao mesmo tempo dispor de seus ambientes de programação preferidos.

Módulo de Consultas: A idéia é construir a expressão da consulta num processo estreitamente ligado ao diagrama E-R, e dirigido pelo sistema.[1] Além de possibilitar facilidades para consulta, essa implementação permite que se tenha uma interface gráfica para sistemas de grande porte.

Esta facilidade permite a inspeção direta da base e navegação pelo esquema, sobre o diagrama, podendo ser útil para o melhor entendimento do significado das construções definidas pelo projetista. Numa fase posterior, esta interface será estendida para um sistema onde filtros possam ser definidos, de modo a automatizar o processo para certas

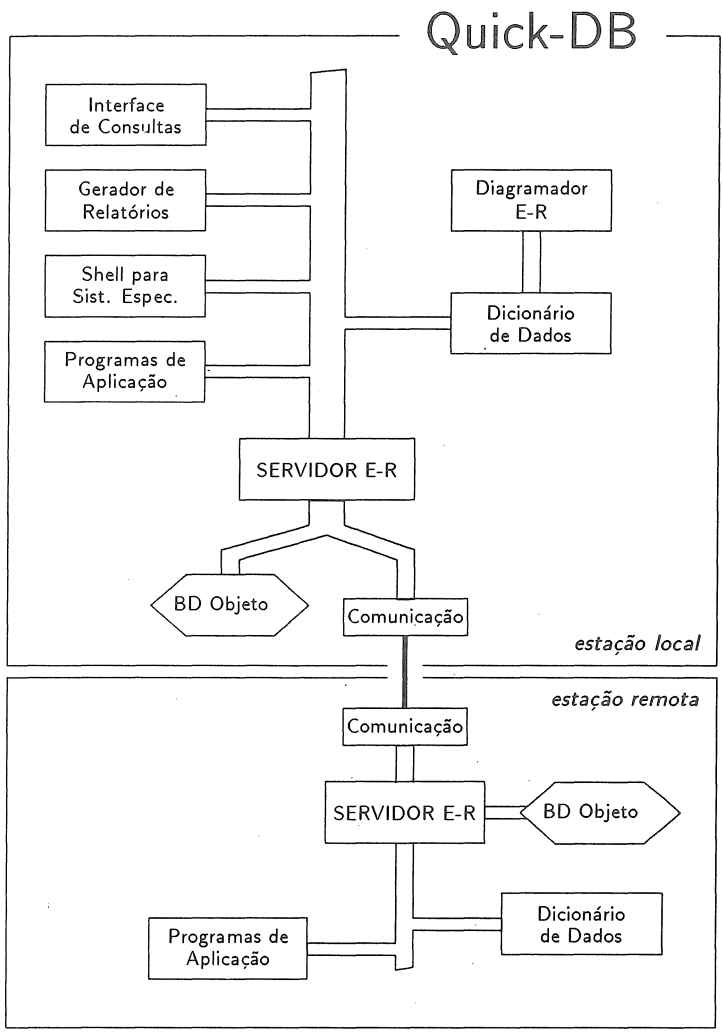


Figura 1. Diagrama funcional

consultas mais simples.

Gerador de Relatórios: Constituído por um conjunto de programas total ou semi-automáticos para geração de relatórios de média e baixa complexidade.

Geradores de Aplicações: O módulo de geração é capaz de produzir programas para os tipos de transação mais usuais como inclusão, exclusão e alteração de entidades e relacionamentos. Nestas operações, são observadas as propriedades do esquema tais como cardinalidades, dependências de inclusão e exclusão, chaves e atributos obrigatórios. Os programas gerados fazem uso da biblioteca de métodos do Turbo Pascal para a definição de interfaces, obtendo-se programas compactos, roustos e com interfaces bastante elaboradas.

Shell para Sistemas Especialistas: Quick-DB dispõe de uma *shell* para sistemas espe-

cialistas baseada no paradigma O-A-V *(object-attribute-value)* para representação do conhecimento.[8] A linguagem para construção de regras abriga elementos como termos e quantificadores, que são acoplados ao servidor de banco de dados de forma natura.
Servidor Remoto: Quick-DB dispõe de um servidor de arquivos que pode resolver referências a bases e arquivos armazenados em estações remotas. Os métodos de acesso podem decidir que determinado acesso deve ser encaminhado a outras máquinas, caso haja indicação a respeito no diretório distribuído. Isso é feito de forma totalmente transparente para os programas, e pode ser configurado independentemente do esquema.

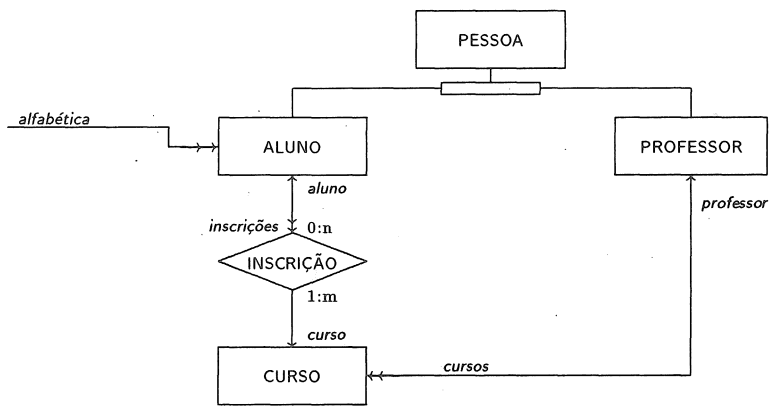


Figura 2. Exemplo simplificado de um diagrama E-R estendido

4. O MODELO ENTIDADE-RELACIONAMENTO E EXTENSÕES.

QUICK-SHELL trabalha com bancos de dados que seguem a disciplina de modelagem do Modelo E-R básico estendido. Esta seção apresenta, de modo resumido, as características do modelo resultante. Maiores detalhes são apresentados em outros trabalhos relacionados a este projeto.[11,12]

ELEMENTOS BÁSICOS. Os elementos básicos são *Conjuntos de Entidades* e *Relacionamentos*. A cada relacionamento está associado um inteiro, dito ser o grau do relacionamento, e um conjunto de funções papel, que mapeiam cada instância do relacionamento em seus componentes. *Atributos* são funções que mapeiam entidades e relacionamentos em suas propriedades. Os atributos podem ser mono ou multi-valorados, quando retornam valores atômicos ou listas de valores, respectivamente. *Conjuntos de valores* são utilizados para definir os domínios dos atributos.

Restrições de Integridade são basicamente as propriedades do esquema, tais como chaves, entidades fracas, cardinalidades, domínios de atributos e outras expressões que eventualmente devam ser respeitadas quando da instanciação da base de dados.

Este conjunto de elementos para a construção de esquemas E-R é bastante conhecido e aparece fartamente na literatura.

EXTENSÕES. No âmbito do nosso projeto, apenas a estrutura básica do Modelo E-R foi utilizada, uma vez que foram introduzidas várias extensões no sentido de tornar o modelo mais rico do ponto de vista semântico e prático. O Modelo E-R utilizado no âmbito deste projeto foi estendido pelos conceitos descritos abaixo.

O conceito de *generalização*[9] foi implementado de forma simples, visando uma solução robusta e eficiente. Todas as implicações ocasionadas pelo conceito são suportadas, tais como herança de chaves e ordens (vide abaixo). Multipla herança de atributos não é permitida, ou seja, cada subtipo pode estar ligado a apenas um supertipo. No exemplo da Figura 2, ALUNO e PROFESSOR são especializações da entidade PESSOA.

Atributos referenciais são atributos que retornam referências a outras entidades do esquema. Este conceito é importante e extremamente útil, uma vez que elimina, em muitos casos, a necessidade de introduzirmos relacionamentos 1:*n* fictícios, que são melhor expressos com o presente conceito. No exemplo da Figura 2, *professor* é um atributo referencial de CURSO e, quando aplicado a uma instância desta entidade, retorna uma instância de PROFESSOR.

Ordens permitem acesso a conjuntos de entidades ou relacionamentos de tal forma que a seqüência em que as instâncias são obtidas fique previamente estabelecida. Uma ordem pode ser parcial ou total. A ordem é total quando todos os elementos do conjunto de entidades, ou do relacionamento que é objeto da ordem podem ser alcançados através da ordem. Ordens podem ser definidas sobre conjuntos de entidades, relacionamentos ou atributos referenciais. Na Figura 2, a ordem *Alfabética*, definida sobre a entidade ALUNO, é uma ordem total. Todas as instâncias daquela entidade podem ser obtidas pela varredura da ordem. Na mesma figura, a ordem *inscrições* é parcial, porque acessa apenas as instâncias de INSCRIÇÃO associadas a uma determinada instância da entidade ALUNO. Da mesma forma, *cursos* é parcial e definida sobre a relação inversa do atributo referencial *professor*, que tem PROFESSOR como domínio. Ou seja, para um dado professor *p* a ordem *cursos(p)* retorna todo curso *c* tal que *professor(c)=p*.

Entidades associativas são relacionamentos que também podem assumir o papel de entidade participante em outros relacionamentos. Entidades associativas podem ter atributos, funções-papéis, ordens e cardinalidades.

Domínios estruturados podem ser definidos para os atributos de uma forma genérica. Assim, conjuntos de valores podem caracterizar domínios simples ou compostos. No segundo caso, o domínio é formado por conjuntos de tuplas, cujos componentes, por sua vez, podem ter domínios compostos ou não.

Atributos virtuais, que podem ser especificados através de regras de produção ou como expressões.

5. ACESSO AO BANCO DE DADOS: OBJETOS E MÉTODOS.

O acesso à base de dados é feito através de um conjunto de métodos, aplicáveis às classes de objetos que formam a base.[7] O programa que comporta a definição de cada método é gerado automaticamente a partir do diagrama E-R definido para o banco de dados. Das definições de entidades, relacionamentos, ordens e outras características do esquema, o gerador monta todas as rotinas necessárias para acesso a base de dados. Com isso, todos os nomes presentes no esquema têm métodos e/ou atributos correspondentes no programa gerado, o que torna a programação fácil e confortável para quem acessa a base de dados.

Este conjunto de métodos é denominado *Servidor-ER*.

Há basicamente quatro classes de objetos: *banco de dados*, *conjunto*, *ordem* e *elemento*.

A classe banco de dados identifica a base de dados como um todo, e aceita os seguintes métodos básicos:

ABRE preparação para acesso;
FECHA encerramento de acesso;
CRIA geração da base, com instâncias vazias. item DESTROI apaga uma base de dados.

Um objeto da classe banco de dados tem ainda um conjunto de atributos e métodos auxiliares que servem para identificação e controles operacionais.

A classe conjunto representa cada um dos conjuntos de entidades E e relacionamentos R existentes na base de dados. Os seguintes métodos se aplicam:

POPULACAO número de elementos na instância do conjunto;
ABRE preparação para acesso;
FECHA encerramento de acesso;
CRIA geração inicial;
DESTROI apaga um conjunto da base;
PONTEIRO obtenção de ponteiro para a instância de um elemento.

Há ainda um conjunto de métodos de acesso restrito que não são relevantes neste contexto. O método *ponteiro* é utilizado para aceeso a uma instância sem que seja necessária a definição de uma variável, permitindo um forma de navegação implícita pela base de dados.

A classe elemento abriga cada um dos elementos de determinada instância de um conjunto de entidades E ou de um relacionamento R, através da própria classe e de suas subclasses. Os métodos do tipo básico são:

REFERENCIA referência física a um elemento dentro de sua instância;
SUBSTITUTO referência lógica (surrogate) que identifica unicamente um elemento dentro da base.

Para cada conjunto de entidades E e relacionamento R, há um subtipo de elemento especificamente definido, que agrega os atributos de E ou R. Um subtipo de elemento, em geral, suporta os seguintes métodos:

ARMAZENA adição de elemento;
ATUALIZA regravação de elemento;
REMOVE remoção de elemento;
INSTANCIA obtenção da instância de um elemento.

Há ainda alguns métodos que tratam especificamente de elementos sujeitos a generalização e especialização. Nestes casos, observa-se que alguns dos métodos acima podem ou não ser aplicados, dependendo das circunstâncias.

Por fim, a classe ordem trata do acesso à base de dados. Uma instância da classe ordem pode ser imaginada como uma lista de referências a elementos da base. Assim, pode-se ter várias ordens definidas sobre um conjunto de entidades ou relacionamento. Através de cada uma dessas ordens, pode-se acessar os elementos segundo critérios de ordenação que são definidos por ocasião da confecção do projeto lógico do banco de dados. A classe ordem admite os seguintes métodos:

INICIA posicionamento no início da ordem;
PROXIMO obtenção do próximo elemento;

FIM teste de fim de lista;

PROCURA busca de um elemento que contenha atributo igual ou posterior a determinada chave;

ENCONTRA busca de um elemento que contenha exatamente determinada chave.

6. UTILIZAÇÃO.

O paradigma de utilização de QUICK-DB passa, basicamente, por três fases: *a)* definição do esquema, *b)* geração dos módulos servidores e *c)* manipulação da base de dados, através da linguagem hospedeira e das ferramentas associadas.

A definição do esquema se dá graficamente e, na medida em que o usuário vai dispondo as entidades, relacionamentos e conexões, o sistema direciona a complementação das informações necessárias para que fiquem completas as definições. Todo o processo se dá com auxílio de *pop-up menus* e *mouse*.

As dimensões do espaço de desenho são virtualmente infinitas. A limitação desse espaço é um problema sério em muitos diagramadores, pois bases de dados de maior porte podem abrigar um número de elementos gráficos da ordem de dezenas, ou até mesmo centenas. O acesso a uma "lupa" de magnificação regulável permite que o usuário se oriente e se movimente no espaço de desenho com relativa facilidade.

O diagramador oferece, também, facilidades para documentação dos esquemas. Para impressão dos diagramas completos, QUICK-DB trabalha com um sistema de paginação que elimina as desvantagens das "colagens" de papéis. Também são disponíveis opções para obtenção de textos prontos para os formatadores TEX e LaTEX.

Além dessas facilidades, o diagramador é capaz de verificar a correção e consistência dos diagramas.

A geração dos servidores é a fase onde QUICK-DB, a partir das informações do dicionário de dados, gera programas que atuarão como servidores das bases de dados.[2] O servidor E-R é composto pelas definições de uma coleção completa de objetos e métodos que permitem a utilização da base a partir de uma linguagem hospedeira. No momento, utilizamos o Turbo Pascal (a partir da versão 5.5) neste papel, mas a estrutura do sistema permite a migração para outra linguagem (como C++, por exemplo).

Quando o programador preparar programas de acesso à base, basta que invoque a biblioteca de definições e, de modo simples e estruturado, poderá construir os procedimentos necessários. Todos os componentes do diagrama estarão disponíveis, bem como um conjunto de definições auxiliares que facilitam a manipulação dos dados.

Outra forma de acesso se faz através do folheador da base, que permite a verificação das instâncias e a navegação pelas conexões do diagrama, respeitando ordens e relacionamentos.

Também têm acesso à base de dados, através do servidor, os programas gerados pelo gerador de aplicações e pela *shell* para sistemas especialistas. Estes últimos, entretanto, definem bases próprias (usualmente denominadas bases de conhecimento) que constituem uma classe de atributos virtuais a serem incorporados ao esquema.

7. CONCLUSÕES.

O sistema em questão está implementado em Pascal, com aproximadamente 35.000 linhas de código. Uma das premissas que adotamos por ocasião da estruturação deste projeto foi a de que as soluções pudessem ser integralmente implementadas, com resultados práticos imediatos.

As técnicas de implementação adotadas permitem que o sistema seja facilmente transportado para outros ambientes de desenvolvimento, notadamente C++.

Extensões. Entre as extensões já estudadas e planejadas, encontram-se:

- novos tipos de dados manipulados sob o paradigma de orientação a objetos, principalmente tipo texto e figuras geométricas;
- acoplamento com servidores SQL.

Trabalhos Correlatos. A abrangência deste trabalho torna-o, de certa forma, não característico. Há na literatura menção de diagramadores E-R, interfaces com sistemas especialistas;[4,5,6] folheadores, geradores de aplicações, dicionários de dados, de forma geral. O mérito de QUICK-DB é reunir sob um mesmo paradigma de utilização todas essas facilidades, oferecendo aos seus usuários um ambiente completo, uniforme e consistente para o desenvolvimento de aplicações de bancos de dados.

8. REFERÊNCIAS.

1. CZEJDO, B., ElMASRI, R., et al.; *A Graphical Data Manipulation Language for an Extended Entity/Relationship Model*, IEEE Computer v.23, p.26-36, March 1990.
2. DELVAUX, M. C.: Geração de Rotinas Orientadas a Objetos para Manipulação de um Banco de Dados, Projeto Final de Curso, Instituto de Matemática, Rio de Janeiro, 1991.
3. GARDARIN G., VALDURIEZ P.; *Relational Databases and Knowledge Bases*, Addison-Wesley Publishing Company, 1989.
4. JARKE, MATTHIAS, VASSILIOU, YANNIS; Coupling Expert Systems with database Management Systems. In: Reitman, W. (ed). *Artificial Intelligence Applications for Business*, Norwood, NJ:Ablex, 1985, cap. 5 p. 65-85.
5. JUNET, M., FALQUET, G. and LEONARD, M.; ECRINS/86: An Extended Entity/Relationship Data Base Management System and its Semantic Query Language, *Proceedings of the Twelfth International Conference on Very Large Databases*, Kyoto, Japan, 1986.
6. KERSCHBERG, L.; KORTEX: An Expert Database System Shell for Knowledge Based Entity relationship model, *Int. Conf. on the Entity/Relationship Approach*, pp 174-187, 1989.
7. MEYER B., *Object-Oriented Software Construction*, New York, Prentice-Hall, 1988.
8. PACITTI, E.C. and SILVEIRA, P.; Um Sistema de regras de Produção Fortemente Acoplado a Bancos de Dados, Dissertação de Mestrado, in *Anais do VII Simpósio Brasileiro de Bancos de Dados*, Porto Alegre, 1992.
9. PECKHAM, M. and MARYANSKI, F.; Semantic Data Models, *ACM Computing Surveys 20*, 3(September 1988).
10. SILVEIRA, P. M.; A Formalization of The E-R Model, *Proceedings of the IX Conferencia Internacional de la Sociedad Chilena de la Ciencia de la Computation*, Santiago, Chile, 1989.
11. SILVEIRA, P. M.; Procedural Data Manipulation Operations for the E-R Model, *Proceedings of the XVI Conferência Latino Americana de Informática*, Assunção, Paraguay, 1990.
12. TOSO, V.E.F., SILVA, M.F.; Gerador de Banco de Dados do Quickdb. Projeto Final de Curso de Informática - IM/UFRJ Julho de 1990.

Rumo à Portabilidade de Componentes de Diálogo

Marcelo Soares Pimenta
Depto. de Ciências Estatísticas e da Computação - UFSC
Caixa Postal 476 - Florianópolis - SC - Brasil
Cep 88040-900 - Fone: +55 482 319739
e-mail: cec1msp@brufsc.bitnet

Abstract

This paper describes the Canonicus model, an object oriented user interface representation model with two main goals: i) to allow dialog component portability among different user interface tools and ii) to provide abstractions to separate clearly user interface designer and application programmer roles. Canonicus' main caracteristics, including generic and concret classes, and our experience in its implementation are presented.

Keywords: user interface tools, object orientation, user interface representational model.

1. Introdução

Um programa interativo é formado por componente computacional (doravante denominado aplicação) e componente de diálogo (que implementa a interface com o usuário, abreviada IU). O componente de diálogo é, em geral, construído através do uso de uma Ferramenta para Desenvolvimento de IU (abreviada FIU). FIUs são conjuntos de rotinas ou *programas que auxiliam a definição e manipulação de IUs [1].. Exemplos de FIUs são Mac* Toolbox, MS-Windows Development Kit e OSF/Motif. Devido às idiossincrasias das FIUs, o programador de aplicação acaba criando diferentes versões da mesma aplicação: uma para cada FIU.

Além disto, a maioria das FIUs existentes são difíceis de usar pelo programador da aplicação, pois contêm literalmente centenas de rotinas [2]. Estas rotinas devem ser adequadamente escolhidas e combinadas não só para manipular todos os eventos de entrada como também para desenhar e manipular os elementos da IU (menus, janelas, botões, etc.) Este excesso de detalhes e tarefas resulta na indesejável unificação dos papéis do programador da aplicação e do projetista de interfaces, uma vez que, mesmo com o uso de editores e arquivos de recursos, estas FIUs não isolam o programador de informações relativas ao projeto de interfaces.

O objetivo deste artigo é apresentar o modelo Canonicus, um modelo representacional de IUs orientado a objetos que visa permitir:

- a portabilidade do componente de diálogo de um programa interativo entre diferentes FIUs; e
- a definição dos detalhes e tarefas da IU em diferentes níveis de abstração de forma a separar claramente os papéis do projetista de IUs e do programador de aplicação.

Na seção 2, apresenta-se o enfoque adotado para alcançar portabilidade. Na seção 3, destacam-se algumas características principais do Canonicus. A separação entre classes

genéricas e concretas é descrita na seção 4. A seção 5 apresenta alguns aspectos da implementação do Canonicus sobre FIUs comerciais. Finalmente, na seção 6 são tecidas algumas observações conclusivas.

2. Canonicus e portabilidade

Dois enfoques têm sido utilizados para prover portabilidade de programas interativos em relação ao componente de diálogo. Ambos são baseados na uniformização da comunicação entre aplicação e componente de diálogo, de forma que a aplicação use o mesmo protocolo de comunicação para chamar rotinas de diálogo em qualquer ambiente.
O primeiro enfoque é o uso de uma FIU transportável, uma FIU que possui uma implementação em vários ambientes [4] . O segundo é o estabelecimento de uma camada intermediária entre a aplicação e cada FIU nativa em um ambiente. Em verdade, esta camada intermediária provê uma interface comum com a aplicação para as diferentes FIUs e, por isto, é também denominada camada canônica [2]. As chamadas da aplicação à camada canônica são mapeadas para chamadas à FIU nativa subjacente. Deste modo, o "look and feel" do programa é o "look and feel" da FIU subjacente, característica adequada aos usuários habituados à consistência de um determinado ambiente. Um exemplo típico de uso deste enfoque é o sistema comercial XVT (eXtended Virtual Toolkit) [5], uma biblioteca de funções implementada inicialmente sobre a Mac Toolbox e o MS-Windows.

O Canonicus baseia-se no segundo enfoque para suportar aplicações em diferentes ambientes com diferentes FIUs e diferencia-se do XVT por usar o paradigma de orientação a objetos [6] para definição e uso da camada canônica e para a implementação do seu mapeamento para as FIUs nativas.

3. Características do Canonicus

O modelo Canonicus é um modelo de representação de IUs (ver [7] para definição de modelos de representação de IUs) orientado a objetos. Dessa forma, pretende-se uma descrição da comunicação IU-aplicação de maneira mais adequada ao projetista de interface e ao programador da aplicação, diminuindo a dificuldade de definição/manipulação de IUs.

A Mac Toolbox, por exemplo, pode ser considerada orientada a objetos na visao do usuário , enquanto sua comunicação com a aplicação é feita segundo o paradigma convencional de rotinas e funções. Com o Canonicus, toda comunicação entre IU e aplicação é estabelecida via mensagens direcionadas para os (e/ou a partir dos) objetos pertencentes a IU.

A notação do Canonicus é baseada na linguagem orientada a objetos Eiffel [8] e por isto usamos os termos "atributos" e "rotinas" ao invés dos tradicionais "variáveis de instância" e "métodos" da nomenclatura Smalltalk. Definir IUs usando o Canonicus é descrever classes (genéricas e concretas) de objetos de IU, especificando seus atributos, mensagens e rotinas. Consequentemente, para operar sobre a IU definida deve-se instanciar objetos destas classes e manipulá-los de acordo com o seu protocolo de mensagens.

Em [9] , encontra-se um exemplo que contém as definições e o uso de uma classe genérica Janela e duas subclasses concretas Jan-Mac e Jan-MSW, que implementam a classe Janela respectivamente sobre a Mac Toolbox [10] e o MS-Windows [11] . As descrições completas (com todos atributos, rotinas e mensagens) das classes genéricas, classes concretas para a Mac Toolbox e para o MS-Windows estão presentes em [12] .

Usando o Canonicus, o projetista de interfaces define as classes dos objetos que comporão a IU e suas apresentações. Os objetos são descritos por suas propriedades, seus componentes e seus operadores disponíveis. Sua apresentação define cores, formatos, estilos e tamanhos, etc. Usando o Canonicus, o programador da aplicação define quando usá-los e como associá-los, junto com seus operadores, às funções da aplicação.

4. Classes Genéricas e Concretas

Canonicus é um formalismo adequado para separar a definição (em níveis) da IU pelo projetista de interfaces de sua manipulação pelo programador da aplicação pois organiza os elementos de IU (menus, janelas, botões, etc.) em classes de objetos de IU . As classes podem ser genéricas (ou abstratas) ou concretas (ou de implementação). Por definição, uma classe *concreta é sempre subclasse de alguma classe genérica.*

Basicamente, uma classe genérica possui a descrição abstrata dos objetos de IU (suas características, componentes e comportamento) enquanto as suas subclasses concretas possuem a definição dos atributos de apresentação destes objetos (forma, cores, tamanho, etc.) e a implementação de seu comportamento. Para isto, na classe concreta são encapsuladas as estruturas de dados e o código desta implementação.

Na definição das classes genéricas, os atributos podem ser de tipos pré-definidos (integer, string, etc.) ou associados a classes definidas enquanto todas as rotinas são deferidas, i. é, não apresentam descrição de implementação.

Na definição das classes concretas, por sua vez, os atributos podem possuir tipos relacionados `a FIU especifica subjacente (chamados atributos FIU), adicionalmente aos outros tipos. Como uma classe concreta é subclasse de uma genérica, herda todos atributos desta, podendo também redefini-los ("override"). Além disto, todas rotinas possuem implementação associada. Os atributos FIU são necessários para a apresemtação dos objetos e para a parametrização desta implementação.

Rotinas deferidas são rotinas especificadas na classe genérica mas realmente implementadas nas suas subclasses concretas, podendo ser encaradas como a descrição genérica de um grupo de implementações específicas. Por isto, as classes genéricas são também denominadas classes deferidas e não podem ter instâncias, pois não há rotinas da classe para manipulá-las.

A definição da implementação das rotinas deferidas das classes genéricas é feita através do mapeamento de uma rotina para um conjunto (possivelmente unitário) de rotinas de uma FIU, o qual corresponde à implementação das tarefas da rotina. Isto é ilustrado na figura 1.

As rotinas deferidas são a chave para a portabilidade. O programador ***deve*** usar as rotinas deferidas, especificadas no nível abstrato. Assim, os detalhes de manipulação/exibição *dos objetos, encapsulados no nível concreto, são escondidos do programador, que não precisa* tomar conhecimento de qual implementação esta' sendo executada/utilizada pelo projetista de interface.

Esta definição tem por proposta:
a) encapsular no nível concreto as características de apresentação dos objetos de IU;

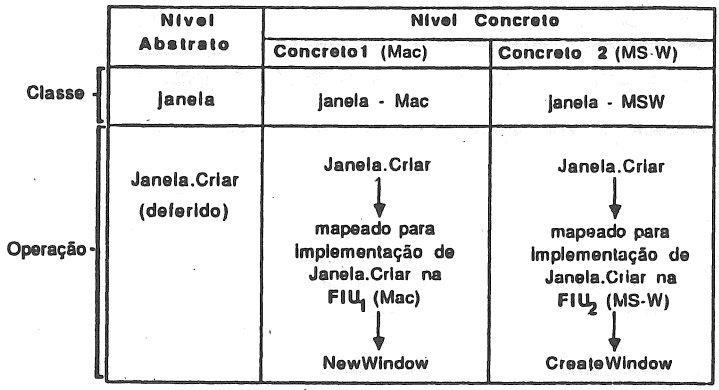


Figura 1 - Implementação de rotinas deferidas: um exemplo.

b) servir para a descrição da implementação das rotinas deferidas das classes genéricas;

c) servir de base para o processo de tradução do Canonicus para uma FIU específica, descrevendo o mapeamento entre eles.

De fato, as classes genéricas formam uma camada canônica abstrata (a FIU Canônica) acima da camada concreta das FIUs, formada pelas classes concretas.

5. Aspectos de Implementação

O mapeamento da camada canônica formada pelas classes genéricas do Canonicus para as FIUs subjacentes (cujas características estão encapsuladas nas classes concretas) é implementado através da tradução de todos seus objetos e mensagens para elementos e rotinas de alguma FIU subjacente (por enquanto a MacToolbox, o MS-Windows Development Kit e o X-View respectivamente em ambiente Macintosh, PC e workstations Sun).

Esta opção por mecanismos de tradução foi tomada por três motivos principais:
- a flexibilidade decorrente do possível uso de vários "look and feel" desde que a tradução do Canonicus para as FIUs que os suportam seja provida;
- a preocupação prioritária com a uniformização da comunicação IU-aplicação como base para a portabilidade de componentes de diálogo;
- a facilidade de experimentação, pois é mais direto construir tradutores do que FIUs.

A tradução pode ser feita segundo dois enfoques: o compilativo e o interpretativo, descritos preliminarmente em [4] . Neste trabalho, por limitações de espaço, falaremos sucintamente da implementação de um tradutor segundo o enfoque compilativo.

Um tradutor que use o enfoque compilativo é tipicamente um pré-processador. Sua função é "varrer"o código fonte do programa interativo e realizar o mapeamento mencionado acima. A partir da especificação das classes genéricas e das concretas para uma FIU específica, o pré-processador deve realizar as seguintes tarefas no programa:

1) Inserir as declarações de variáveis e tipos específicos necessários à execução das rotinas da FIU específica;

2) Substituir as chamadas/invocações das operações/rotinas do Canonicus pelos trechos de código relativos à implementação destas rotinas na FIU particular.

Após a execução da tarefa 1, o conjunto das declarações do programa é a união das declarações que já existiam no programa com as variáveis e estruturas de dados para uso das FIUs específicas.

Após a execução da tarefa 2, o programa pode ser submetido a um compilador.

A figura 2 apresenta um esquema do processo de tradução:

i) O Programa Interativo contém, nos pontos de programa em que a IU deve ser acionada, mensagens aos objetos do Canonicus (chamada à FIU Canônica).

ii) O protocolo genérico de mensagens do Canonicus (camada canônica) deve ser traduzido pelo pré-processador para rotinas de interação de uma FIU específica, substituindo-se qualquer referência a elementos do Canonicus por referências a elementos presentes nas FIUs específicas.

iii) O pré-processador gera um programa interativo para uma FIU determinada (programa para FIU) com as devidas definições, chamadas ou corpos de procedimentos necessários para que a FIU execute o diálogo especificado.

iv) Como as chamadas do componente de diálogo (tanto ao Canonicus quanto à FIU específica) estão embutidas num programa, o passo seguinte é submeter este programa para FIU ao compilador disponível para que se obtenha o código executável do programa (Código Executável FIU). Através da execução deste código, o usuário final interage com o programa interativo.

O enfoque compilativo realiza todo mapeamento ***a tempo de compilação***. A cada alteração das interações ou dos pontos de interação do programa este deve passar pelos passos i a iv acima.

Atualmente está em estágio final de desenvolvimento um pré-processador que mapeira o componente de diálogo do Canonicus para a Mac Toolbox e espera-se que, em breve, os pré-processadores para o MS-Windows e o X-View também estejam funcionando.

6. Conclusões

Este trabalho descreveu as principais características do modelo Canonicus, que visa propiciar a portabilidade de componentes de diálogo de programas interativos desenvolvidos sobre algumas FIUs comerciais e diminuir a dificuldade de definição/manipulação de IUs.

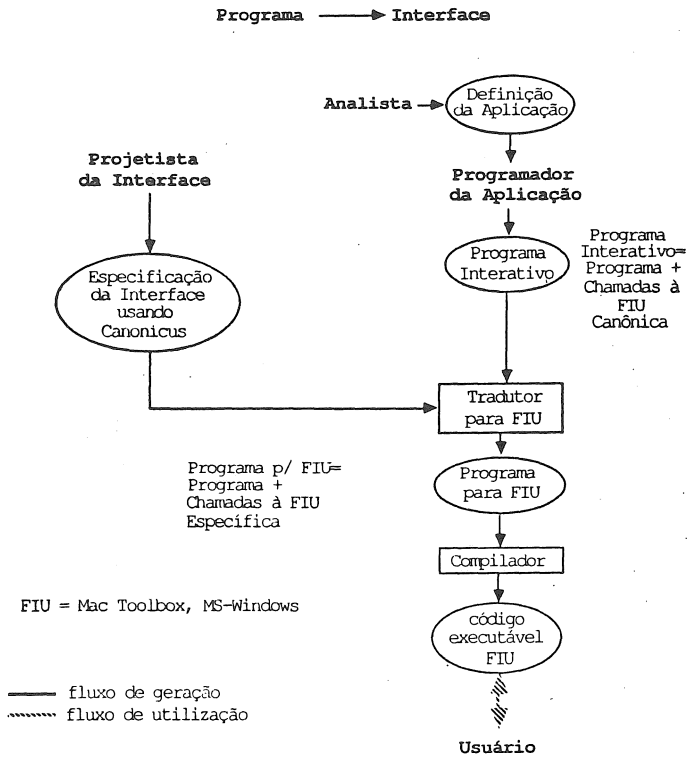


Figura 2 - Esquema do processo de tradução (enfoque compilativo)

O Canonicus é uma camada orientada a objetos padronizada entre as FIUs e a aplicação e é implementada através de mecanismos de tradução, com seus objetos e operações mapeados para elementos e operações das FIUs subjacentes. Desde que classes genéricas do Canonicus podem ser acessadas somente através das operações associadas a elas, os usuários das classes genéricas não tem acesso às especificidades das classes concretas. Isto provê um completo encapsulamento das peculiaridades das FIUs subjacentes e torna o uso de classes genéricas adequado para o programador.

Outra vantagem do Canonicus é a reusabilidade. O enfoque orientado a objetos adotado possibilita definir novas classes/componentes de IU a partir das classes primitivas existentes, que provêm um conjunto de objetos que podem ser usados como "blocos de construção", já devidamente testados e depurados.

O Canonicus pode servir como suporte a uma metodologia de desenvolvimento de IUs, com a separação dos papéis do projetista de interfaces e do programador de aplicações e com suas hierarquias de classes propiciando projeto de IUs com níveis diferenciados de detalhes. O uso de uma mesma notação para especificação (classes genéricas) e implementação (classes concretas) de IUs evita os erros decorrentes da tradução de um formalismo de projeto para um de implementação e beneficia as tarefas dos usuários projetistas, mais notadamente o programador, que pode usar a IU sem se preocupar em conhecer ou fornecer os detalhes para sua execução.

7. Bibliografia

[1] MYERS, B. "User Interface Tools: Introduction and Survey" IEEE Software, Jan 1989.

[2] MYERS, B. et alli "Garnet: Comprehensive Support for Graphical Highly Interactive User Interfaces". IEEE Computer Nov 1990.

[3] SZCZUR, M. "Transportable Applications Environment - An Integrated Design-to-Production UIMS", Ninth Annual Conference and Exposition to Computer Graphics Applications, Proc., Mar 1988.

[4] FRAINER, A.S. ; PIMENTA, M.S.; PRICE, R.T. "Como Obter Portabilidade de Programas Interativos" Anais X Congresso SBC, Vitória, Julho 90.

[5] VALDES, R. "A Virtual Toolkit for Windows and the Mac", BYTE Mar 1989.

[6] GIRARDI,M.R. ; PRICE, R.T. . "O Paradigma de Desenvolvimento por Objetos" Revista de Informática Teórica e Aplicada, V. 1. N. 2, 1990, pp 69-98.

[7] HARTSON, H.R.; HIX, D. "Human-Computer Interface Development: Concepts and Systems for its Management" ACM Computing Surveys V.21 n. 1 Mar 1989.

[8] MEYER, B. "Object Oriented Software Construction", Prentice-Hall, Series in Computer Science, 1988.

[9] PIMENTA, M.S.; HEUSER,C.A. "Canonicus: Um Modelo para Portabilidade de Programas Interativos". Anais do V Simpósio Brasileiro de Engenharia de Software, Ouro Preto, Out 1991.

[10] Apple Computer "Inside Macintosh V. I,II,III", Addison-Wesley, 1985.

[11] JAMSA, K. "Windows Programming Secrets", Osborne McGraw-Hill, Berkeley, USA. 1987.

[12] PIMENTA, M.S. "Um Modelo Canônico de Ferramenta para Desenvolvimento de Interface com o Usuário",
Dissertação de Mestrado. CPGCC-UFRGS. Jan 1991.

UM MODELO PARALELO PARA O APRENDIZADO EM REDES NEURONAIS BASEADO EM ALGORITMOS GENÉTICOS

Mauricio Solar[1/2]
Sueli B. T. Mendes[2]

[1]USACH: Universidad de Santiago de Chile
Depto. de Ingenieria Informática - Casilla 10233-STGO
Avda. Ecuador 3659 - Estación Central - Santiago de Chile
FAX: (562) 7763511 e-mail: MSOLAR@USACHVM1.BITNET

[2]COPPE/UFRJ
Programa de Engenharia de Sistemas e Computação
Centro de Tecnologia - COPPE - UFRJ - Caixa Postal 68511
Rio de Janeiro - RJ - CEP 21945 - Brasil

RESUMO

Na ciência da computação onde os algoritmos seriais são usualmente paralelisados com a ajuda de truques e contorções, não deixa de ser uma ironia que os Algoritmos Genéticos (AGs) (altamente paralelos) são efetuados em forma serial com os mesmos truques pouco naturais e sem elegância. O artigo apresenta um modelo paralelo para um Algoritmo Genético Conexionista (AGC) mapeado naturalmente sobre uma máquina paralela (iPSC da Intel e Rede de Transputers da INMOS) com bons resultados em termos de tempo de comunicação, tempo de processamento e tempo de resposta. A implementação do modelo AGC mostra que é adequado para tarefas que operam num meio ambiente dinámico. O modelo é baseado nos mecanismos de busca dos AGs e em resultados provenientes das Redes Neuronais Artificiais (RNAs). As propriedades e os parâmetros do AGC permitem ao sistema se adaptar às mudanças do meio ambiente, respondendo adequadamente às situações para as quais não foi treinado. O AG é capaz de superar as limitações de aprendizado automático das RNAs, permitindo alterar o conhecimento na medida que o meio ambiente muda. O AGC apresenta propriedades desejáveis de ambos os campos, ou seja capacidade de generalização das RNAs e capacidade de adaptação dos AGs. O AGC opera com uma familia (população) de RNAs (indivíduos) com arquitetura idêntica, cujo conhecimento armazenado nos pesos é diferente.

1.- INTRODUÇÃO

As Redes Neuronais Artificiais (RNAs) [HOPFIELD, 82; PAO, 89] têm duas fases de operação: aprendizado e consulta. No aprendizado, a RNA constrói representações internas complexas de seu meio ambiente, sendo este um dos maiores objetivos da pesquisa de RNA (obter procedimentos eficientes de aprendizado [MICHALSKI et alii, 86]).

O algoritmo de aprendizado descobre os pesos que fazem a RNA apresentar o comportamento desejado, com base num conjunto de exemplos. O algoritmo deve ser capaz de modificar os pesos das conexões de forma que as unidades internas que não são parte da entrada ou da saída venham a representar aspectos importantes do domínio do problema. O conjunto de exemplos (o qual não é simples de determinar), geralmente não representa todos os padrões da entrada, o qual piora com um meio ambiente dinámico (mais complexo), e na consulta, a RNA não é capaz de responder as mudanças do meio ambiente [MUHLENBEIN, 90]. Esta limitação é produto da dificuldade destas em capturar todos os padrões no aprendizado [HINTON, 89]. Daqui surge a necessidade de dispor de um sistema que possa se adaptar às mudanças do meio ambiente.

Várias propostas têm sido apresentadas como uma solução a este problema. Uma delas é o uso dos Algoritmos Genéticos (AGs) [DeJONG, 80; GREFENSTETTE, 86; HOLLAND, 86; DAVIS, 87; BOOKER et alii, 89; GOLDBERG, 89;], os quais têm mostrado ser uma boa alternativa pela sua flexibilidade na adaptação. Estes AGs "puros" não possuem um mecanismo de aprendizado eficiente que oriente a busca, dado que sua forma de avaliar as soluções (função objetivo) é complexa de obter. Outro problema que apresentam estes AGs "puros" é o tamanho inicial da população [GOLDBERG, 85], o qual chega a ser tão grande, que é inviável a sua implementação pelo espaço de memória e tempo de processamento necessários [ROBERTSON, 88].

Uma das soluções proposta em SOLAR (92) e apresentada neste trabalho é um modelo de Algoritmo Genético Conexionista (AGC) [OOSTHUIZEN, 87; MONTANA & DAVIS, 89; WHITLEY et alii, 90; PALAGI, 91; MACHADO et alii, 92] que combina ambas as técnicas, onde é possível modificar os pesos das RNAs de tal forma que o conhecimento representado nas conexões, seja alterado na medida que o meio ambiente exija. Isto permite herdar as propriedades desejáveis de ambas as técnicas.

Muitos componentes do modelo do AGC apresentado, incluindo as RNAs, ajuste dos pesos e mecanismos de aprendizado são inherentemente paralelos [HOLLAND, 86; PETTEY et alii, 87; ROBERTSON, 87]. Isto indica que uma implementação paralela deste modelo pode ser mapeado em forma natural sobre uma arquitetura paralela.

Uma questão importante que está em aberto no estudo dos AGs é o tamanho inicial ótimo da população. O problema está no compromiso entre o montante da busca genética que pode ser efetuada e o montante disponível de tempo real. Se a população é muito pequena, então é possível que o AG não ache uma boa solução pela insuficiencia de esquemas na população [BOOKER, 87]. Se a população é muito grande, é possível que o tempo usado para efetuar todas as avaliações seja inaceitável [ROBERTSON, 88]. Uma implementação paralela do AGC pode resolver o compromiso entre busca genética e tempo real. O AGC paralelo opera sobre uma grande população de vários grupos de indivíduos distribuidos num multiprocessador.

BETHKE (81) calculou várias estimações de complexidade para um mapeamento particular de AGs em máquinas paralelas. Conclui que o cálculo da adaptabilidade da média da população é o principal gargalo serial em implementações de AGs desses tempos. Mas não simulou nem implementou um AG paralelo. GR\EFENSTETTE (81) examinou várias implementações paralelas de AGs, classificando quatro protótipos: (1) Mestre-Servidor síncrono; (2) Mestre-Servidor semi-sincrono; (3) Concorrência assincrona distribuída; e (4) Redes.

HOLLAND (86) mostra a natureza paralela do paradigma reprodutivo e a eficiência inherente do processamento paralelo. Mostra um tipo de computador celular para mapear a reprodução.

RIOLO (86) introduz o primeiro AG em aprendizado (Sistema de Classificação), o qual ROBERTSON (87) implementou na "Connection Machine" [HILLIS, 86]. PETTEY et alii (87) apresentam um AG paralelo que consiste de um grupo de nós de AG idênticos. Cada nó do AG mantêm uma pequena população, e em cada geração, cada nó se comunica com os seus vizinhos. Esta comunicação consiste em enviar os melhores indivíduos da população local para os vizinhos e recever os melhores indivíduos de cada vizinho, inserindo os novos indivíduos na população local.

GOLDBERG (89) propõe dois procedimentos paralelos orientados a objetos para os AGs: a "Comunidade" e a "Planta de Polinização". Sendo cada subprocesso um objeto, é simples isolar o processamento, o requerimento de memória e a largura de banda da comunicação requerida para cada objeto. Esta perspectiva pode permitir um mapeamento mais eficiente dos AGs em computadores paralelos.

O trabalho define os elementos básicos do AGC, suas características, princípios e parámetros. Depois mostra as características do modelo paralelo em termos dos parámetros do modelo, tais como tamanho da população do AG, ordem da RNA e comunicação necessária. No fim apresenta as conclusões do modelo paralelo implementado no hipercubo iPSC da Intel e numa Rede de Transputers da INMOS.

2.- CONCEITOS GERAIS

O modelo foi implementado num sistema de multiprocessamento com memória distribuída, sob um esquema de computação MIMD [SOUCEK & SOUCEK, 88]. As memórias e processadores estão conectados por uma rede de interconexão ponto-a-ponto; descrita em termos de nós e enlaces (do tipo "full-duplex"). Cada processador pode usar todos os seus enlaces ao mesmo tempo (comunicação limitada-pelo-enlace), porque é o número de enlaces os que limitam a comunicação [seleção do sistema multiprocessador aparece em SOLAR (92)].

Os multiprocessadores que compartilham memória têm dificuldade no acesso concorrente à memória. No entanto, se a memória é também distribuída, a transferência de dados depende da topología, para o qual é necessário escolher a topología. O esquema MIMD permite diferentes tipos de topologias a serem usadas. Na escolha da topología qualquer decisão é um compromiso entre custo e desempenho [HILLIS, 86], pelo qual tem-se usado diferentes tipos de topologías: hipercubo, anel, grade, etc. O modelo apresentado baseia-se na topología de Hipercubo, na qual é facil mapear outras topologias, tais como uma grade, uma árvore, etc.

Os processadores do hipercubo se comunicam pela troca de mensagens, mas o esquema de comunicação depende do problema. No modelo desenvolvido são usados os esquemas de comunicação de Difusão e Convergência [DINNING, 89] para sinconizar os processos. O tempo de comunicação T para enviar uma mensagem desde um processador para um vizinho é modelado como constante. Muitas pesquisas mostram que o custo elemental T depende do comprimento L da mensagem. No AGC desenvolvido a grande maioria das mensagens são curtas (1 byte), para o qual se usou o modelo constante (T=1) ao inves do modelo linear. No modelo constante as mensagens podem ser particionadas e recombinadas sem mudar o tempo de propagação.

2.1.- Arquitetura do Hipercubo H_d [FOX, 88]

Um hipercubo H_d é um grafo com $N=2^d$ vértices. Cada vértice é marcado por um número binário de d-bits. As arestas estão entre dois vértices cuja marca difere em precisamente um bit. O hipercubo pode ser definido recursivamente como: um hipercubo de 1-dimensão é uma aresta com um vértice marcado 0 e outro vértice marcado com 1. Um hipercubo de (d+1)-dimensões é construido apartir de dois hipercubos de d-dimensão, H_d^0 e H_d^1, pela soma de arestas desde cada vértice em H_d^0 para vértices em H_d^1 que possuem a mesma marca e logo prefixando todas as marcas em H_d^0 com um 0 e todas as marcas em H_d^1 com um 1. O hipercubo é usado em vários computadores paralelos (iPSC da INTEL) por seu pequeno diámetro $D=d=\log_2 N$ e pelo seu pequeno grau $\triangle=d=\log_2 N$. Desafortunadamente, possui $(N \log_2 N)/2$ arestas.

3.- IMPLEMENTAÇÃO PARALELA DO ALGORITMO GENÉTICO CONEXIONISTA

No modelo AGC tem-se N ($N=2^d$) AGs locais e independentes com memória distribuida, operações genéticas e funções objetivos independentes. Os N processos operam normalmente, com a exceção que os melhores indivíduos de cada geração são difundidos para as outras subpopulações através da rede de comunicação. Com uma comunicação relativamente intermitente, o largo de banda das ligações é reducido se comparado com outros esquemas. A confiança deste modelo é alta, pela autonomia dos processos independentes.

3.1.- Suposições do Modelo AGC

O AG opera com uma população de indivíduos, que representa as soluções do problema no qual está imerso. O modelo de AGC tem como população uma familia de RNA com uma arquitetura fixa. Cada RNA é representada como uma cadeia, cujos componentes correspondem aos pesos das conexões entre os neurônios. O AGC opera sobre esta população escolhendo aqueles que melhor se adaptam à entrada proveniente do meio ambiente, gerando uma saída. Se na população de RNA não existir nenhuma RNA que responda adequadamente, então os operadores genéticos combinam algumas destas RNAs para obter novas soluções (com idêntica arquitetura, mas com pesos diferentes) e elimina os piores individuos da população (substituição elitista). O modelo do AGC se concentra nas seguintes suposições:

- O meio ambiente, as entradas e saídas podem ser representadas por uma RNA de três camadas: nós de entrada, nós escondidos e nós de saída. Cada RNA corresponde a uma cadeia de símbolos de um comprimento fixo (L bits) sobre um alfabeto A. Este alfabeto binário é {0,1}. No modelo de AGC cada cadeia representa uma arquitetura de RNA com seus pesos;
- Cada ponto no espaço do problema (espaço total: $N=2^L*2$) pode ser considerado como uma combinação de pesos de uma RNA representado unicamente dentro do sistema por uma cadeia gerada do alfabeto A. Nesta cadeia (cromossoma) do modelo de AGC cada posição específica (locus) representa um bit da conexão da RNA, e seus valores (alelos) indicam um bit do peso de uma conexão;
- O sistema mantém uma população P(t) fixa de RNAs em cada processador representando as soluções atuais do problema;
- O tempo é medido em intervalos discretos chamado gerações.

O AGC conhece as regras para executar a tarefa específica, a qual é definida por exemplos apresentados ao sistema, e a função objetivo determina o desempenho do sistema. Cada RNA tem associado um peso (dado pela função objetivo) que indica a eficiência dela em executar a tarefa desejada.

3.2.- Princípios do Modelo AGC

O AGC usa a familia de RNAs como representação, e a estrutura do conceito é modelada pela organização, variabilidade e distribuição dos pesos entre os neurônios de cada RNA. O modelo se baseia nos seguintes princípios:

Codificação do Cromossoma (Indivíduos): Um indivíduo é uma RNA com uma topología definida e com conhecimento armazenado nos pesos das conexões. Uma conexão da RNA é codificado como um número real de 8 bits (-12.7 até 12.8). Cada indivíduo do AGC é composto por uma cadeia de comprimento fixo (L), onde a cadeia representa as conexões entre os nós de entrada, os nós de saída e os nós escondidos da RNA. Uma posição nesta cadeia (*locus*) corresponde a um bit da conexão entre dois neurônios (nó). O valor armazenado nesta posição (*alelo*) é um número binário tomado do alfabeto {0,1} que indica um bit do peso da conexão.

Uma representação cromossómica que obedeça à hipotese dos blocos de armar garante a busca do ótimo da solução [BOOKER et alii, 89]. Os blocos de sinapses que têm efeito sobre um único neurônio da camada de saída devem ser contiguos aos outros blocos que tem influencia sobre o mesmo neurônio. Isto indica que a estrutura deve conter contiguamente todos os blocos importantes ao aprendizado de cada neurônio de saída.

Seja uma RNA do tipo "feedforward" com três camadas, onde a camada de entrada possui l neurônios N_i, $1<=i<=l$, a camada escondida possui m neurônios N_j, $1<=j<=m$, e a camada de saída possue n neurônios N_k, $1<=k<=n$.

Um bloco de sinapses é uma estrutura criada com o objetivo de reunir coerentemente as sinapses que influenciam um determinado neurônio de saída. Define-se o bloco de sinapses que influencia o neurônio N_k da camada de saída conectado ao neurônio N_j da camada intermediária pela lista resultante da concatenação da sinapse que une N_j a N_k com todas as sinapses que convergem para N_j [PALAGI, 91]:

$$\text{Bloco}(k,j) = [w_{kj}, [w_{ji}]_{i=1}^{t}]$$

onde o simbolo $[w_{ji}]_{i=1}^{t}$ significa a concatenação dos valores A_i, $1<=i<t$.

A escolha de uma RNA "feedforward" se baseia principalmente na existência do Teorema de KOLMOGOROV sobre o Mapeamento em Redes Neuronais, mostrado a seguir e cuja prova está em HECHT-NIELSEN (87):

TEOREMA DE KOLMOGOROV: Dada qualquer função contínua $f:[0,1]^n \longrightarrow R^m$, $f(x)=y$, f pode ser implementada exatamente por uma RNA "feedforward" de três camadas, tendo n elementos de processamento na primeira camada (x-entrada), (2n+1) elementos de processamento na camada intermediária, e m elementos de processamento na última camada (y-saída) [HECHT-NIELSEN, 90].

A existência deste teorema indica que a representação escolhida além de cumprir a teoria dos blocos de armar, permite mapear qualquer função contínua sobre a arquitetura de RNA no cromossoma do AGC. O comprimento L do cromossoma é dado por:

$$L = (n * m * (1 + 1) * 8) \text{ bits}$$

Segundo o comprimento do cromossoma binário é possível determinar o tamanho, ou melhor a complexidade do problema como:

$$N = 2^L * 2$$

Função Objetivo: A função objetivo para o modelo AGC deve guiar o aprendizado da RNA na direção do erro mínimo das suas saídas, quando existe um reforço que vem do meio ambiente externo. Para possibilitar a solução do aprendizado com AGs faz-se necessário definir uma função objetivo de forma que seu máximo represente o menor erro de aprendizado. Pode-se definir o erro de aprendizado de uma RNA como o erro médio quadrático entre a resposta efetiva (o) e a resposta desejada (t) de cada neurônio N_k da camada de saída somado para cada um dos v padrões a serem aprendidos [PALAGI, 91]. Ou seja:

$$E = \sum_{p=1}^{v} \sum_{k=1}^{n} 1/2(t_{pk} - o_{pk})^2$$

Para normalizar o erro, pode-se dividir pelo máximo erro possível nv/2 do aprendizado dos v padrões, obtendo-se o erro médio, sendo a média tomada sobre os n neurônios e sobre os v padrões:

$$E_N = 2E/nv$$

A função objetivo deve maximizar a solução, para o qual se usa o inverso do erro de aprendizado (o qual deve ser minimizado), estabelecida por:

$$f(E_N) = 1/(E_N)^c$$

onde c tem o papel de controlar a intensidade desta relação.

Heurísticas de Aprendizado: No AGC a adaptação de um conjunto fixo de RNAs é feito para que elas respondam melhor à tarefa desejada. Isto é efetuado pelo AG criando novas RNAs da mesma arquitetura (e pesos diferentes) e é executado pelos mecanismos de heurística dos Operadores Genéticos.

3.3.- Parâmetros do Modelo de AGC

É possível elaborar vários tipos de AGC envolvendo variações, tais como probabilidade de aplicar os operadores genéticos, tamanho fixo ou variável da população, percentagem da população sujeita a mudança, etc. A seguir são apresentadas as características dos parâmetros do AGC [GREFENSTETTE, 86]:

Tamanho da População N (TP): GOLDBERG (85) desenvolveu uma teoría para o ótimo do TP para AGs com código binário baseada no comprimento do cromossoma. Para um cromossoma com L=27, o TP ótimo é 77. Um cromossoma com L=27 representa uma RNA extremamente pequena (menor que uma RNA de 1X2X1 neurônios). Com um L=60, o qual ainda representa uma RNA muito pequena (2x2x2), o TP ótimo é 10.200, que é muito grande. Para uma RNA de tamanho médio é necessário um cromossoma de L=500, cujo TP é excessivamente grande, o qual não permitiria um processamento em tempo real. A teoría de GOLDBERG foi desenvolvida baseada num AG puro, ou seja o problema a ser resolvido é único. No caso das RNAs a quantidade de subproblemas a serem resolvidos depende da quantidade de padrões a serem apresentados à RNA. Portanto, por cada padrão novo, o TP ótimo deve incrementarse proporcionalmente. Se se desejam ensinar 4 padrões numa RNA com L=60, o TP ótimo é 4*10.200=40.800, o qual piora ainda mais o processamento em tempo real.

Baseado no trabalho de ROBERTSON (88), no modelo paralelo do AGC, é possível mostrar o seguinte: no modelo sequencial do AGC um indivíduo é selecionado baseado no seu desempenho na população completa, mas no modelo paralelo um indivíduo é selecionado baseado no seu desempenho na população local. Esta diferença mostra três aspectos interessantes de antecipar: (1) O modelo paralelo do AGC é independente do TP, ocupando pouco espaço de memória; (2) A convergência do modelo é mais rápida, permitindo sua aplicação em tempo real; (3) Produz melhores resultados, dado que intercambiam os melhores indivíduos locais entre as diferentes populações.

O modelo implementado usou um TP=50 indivíduos, independente da quantidade de processadores usados.

Procedimento de Inicialização: Os pesos das RNAs da população inicial são escolhidos em forma aleatória com uma distribuição de probabilidade dada por $e^{-|x|}$. Isto é diferente da distribuição de probabilidades inicial dos pesos usualmente usados em "backpropagation", a qual é uma distribuição uniforme entre 0.0 e 1.0. A função de distribuição de probabilidades usada é o refleto das observações empíricas dos pesquisadores de que as soluções ótimas tem tendência de conter pesos com valores absolutos arbitrariamente grandes [MONTANA e DAVIS, 89].

Escolha dos Pais: A escolha dos pais é feita a partir da população ordenada em forma ascendente (proporcional à função objetivo). De acordo com o valor da função objetivo são escolhidos pares de pais em forma probabilistica para gerarem novos filhos. Aquelas RNAs com melhor desempenho têm maior probabilidade de serem selecionadas como pais da seguinte geração.

Probabilidade do Operador

Razão de Cruzamento C (RC): Em cada população nova, são cruzados C*N indivíduos. No AGC se efetuaron varios testes com a RC, usando os valores 0.5 até 0.9 com incremento de 0.1;

Razão de Mutação M (RM): ocorrem aproximadamente M*N*L mutações por geração. O valor de M no AGC foi escolhido empiricamente igual a 0.04;

Intervalo de Geração G (IG): a percentagem da população que será substituida em cada geração, N*G indivíduos de P(t), é variável dependendo da adaptabilidade da população. A adaptabilidade é determinada pela soma do valor da função objetivo para todas as RNAs, não excedendo o 50% da população;

Fator de Graduação W (FG): No AGC é importante manter a diversidade da população, para o qual o FG usado pode chegar até 0.5 da população sem perda de informação;

Estratégia de Seleção S (E$): O AGC usa a ES elitista; primeiro efetua uma ES pura e logo os com melhor desempenho são escolhidos para sobreviver na próxima geração, permitindo modificá-los por cruzamento, mutação ou erro de mostra.

3.4.- Operadores Genéticos

Com o objetivo de estudar o comportamento dos diferentes operadores nas diferentes situações, foram usados um grande número de operadores genéticos, selecionando o melhor conjunto de operadores para a versão final do modelo. Os operadores podem ser agrupados em duas categorias básicas: mutação e cruzamento. As variações do operador de mutação são: mutação tendenciosa, mutação não tendenciosa, mutação de nós e mutação dos nós fracos. As variações do operador de cruzamento são: cruzamento dos pesos e dos esquemas. A seguir descreve-se as variantes dos operadores que foram usados:

Mutação tendenciosa: para cada entrada do cromossoma, o operador substitui com uma probabilidade fixa p=0.04 um valor aleatório escolhido da distribuição de probabilidades de inicialização;

Mutação não tendenciosa: para cada entrada do cromossoma, o operador soma com uma probabilidade fixa p=0.02 um valor aleatório escolhido da distribuição de probabilidades de

inicialização. A mutação tendenciosa é melhor que a não tendenciosa pela seguinte razão: os pais são escolhidos com a tendencia de serem melhores que a média, portanto, os pesos dos pais têm tendencia de serem melhores que uma escolha aleatória. Usando uma distribuição de probabilidades que tem tendência pelos valores presentes dos pesos dá melhores resultados que uma distribuição de probabilidades centrada em zero;
Mutação de nós: este operador seleciona n nós de não entrada da RNA. Para cada conexão de entrada nestes nós, o operador soma ao peso da conexão um valor aleatório tomado da distribuição de probabilidades de inicialização. Então codifica a nova RNA no cromossoma filho;
Mutação dos nós fracos: o conceito de peso dos nós é diferente do conceito de erro usado em "backpropagation". Por exemplo, um nó pode ter erro zero se todas as suas conexões de saída estão em zero, mas este nó não está contribuindo nada positivo para a RNA e não é um nó forte. A definição de peso de um nó escondido numa RNA "feedforward" é a diferença entre a avaliação da RNA intacta e a avaliação da RNA com aquele nó lobotomizado (com suas saídas em zero).

O operador de mutação de nós fracos calcula os pesos de cada nó escondido de uma RNA. Logo seleciona os m nós mais fracos e realiza a mutação em cada uma das suas conexões de saída e de entrada. Esta mutação não é tendenciosa se o peso do nó é negativo e é se o peso do nó é positivo. Logo codifica a nova RNA no cromossoma como um filho.

Cruzamento dos pesos: este operador coloca um valor dentro de cada posição do cromossoma filho selecionando aleatoriamente um dos pais e usando o valor na mesma posição que o cromossoma pai;
Cruzamento dos esquemas: os diferentes nós numa RNA tem diferentes papeis. Para uma RNA com camadas totalmente interligada, o papel de um nó dado pode depender só da camada onde se encontra e não da posição dentro da camada. De fato, pode-se trocar o papel de dois nós A e B na mesma camada de uma RNA como segue: passa por todos os nós C ligados (pela entrada ou saída) com A (portanto com B). Intercambiar os pesos das conexões entre C e A com aquele da conexão entre C e B. Ignorando a estrutura interna, a nova RNA é idêntica com a anterior (dada uma entrada, a saída é exatamente a mesma).

Os filhos produzidos pelos operadores de cruzamento acima mencionados afeta grandemente as estruturas internas dos pais. O operador de cruzamento de esquemas reduz esta dependencia da estrutura interna fazendo o seguinte: para cada nó no primeiro pai, tenta achar um nó no segundo pai, o qual tem o mesmo papel observando o número de entradas de ambas as RNAs e comparando a resposta dos diferentes nós. Então combina o segundo pai de tal forma que os nós que tem o mesmo papel estejam na mesma posição.

4.- OPERAÇÃO PARALELA do Modelo AGC

O modelo AGC foi implementado na linguagem C para o hipercubo iPSC [INTEL CORP., 88], e na linguagem OCCAM para uma rede Hipercubica baseada em Transputers. Ambas as implementações deram bons resultados em termos de tempo de comunicação, sendo que na Rede de Transputers é necessário realizar o controle das mensagens enviadas entre os processadores. No iPSC é possível explorar as primitivas de comunicação proporcionadas pela máquina, tendo-se uma implementação mais simples de sincronizar. A operação do modelo é resumida no seguinte algoritmo:

```
ALGORITMO AGC;
INICIO
  Inicialização; (O Mestre inicializa cada nó Servidor)
  REPETIR
    Recebe Entrada; (O Mestre envia a Entrada aos Servidores)
    Transmite Entrada para os AGs Locais;
    REPETIR em paralelo para cada AG Local
      PARA cada RNA REALIZAR
       INICIO
         Dominio AG --> Dominio de RNA;
         Testar RNA com Padrão de Entrada;
         Obter Erro Quadratico Médio;
         Obter Valor da Função Objetivo;
       FIM;
      Ordenar População segundo Desempenho;
      Aplicar Operadores Genéticos;
      Individuo de Melhor Desempenho --> Piores Individuos;
      Transmitir Melhor Individuo ao Processo Mestre;
      Receber Melhor Individuo do Processo Mestre;
      Inserir Individuo recebedo na População Local;
    ATÉ Convergência Local;
    Aplica Saída;
  ATÉ Fim do Processo;
FIM.
```

A Inicialização consiste em adquirir os dados iniciais do problema e inicializar cada população local do AGC. Segundo a ordem da RNA definida pelo usuario é calculado o comprimento do cromossoma e sua representação no domínio do AGC no processo Mestre, o qual envia os dados de inicialização para os processos locais. O Mestre adquire os dados do meio ambiente (o processo indica uma referência que determina a direção do controle). A entrada é considerada como um padrão que a RNA deve aprender. Logo o Mestre envia a entrada para os processos locais (Recebe Entrada).

Dado que o AG opera com uma representação cromossómica da RNA é necessário leva-la para o dominio das RNAs, no qual se realiza a consulta da RNA para verificar o erro médio desta com os padrões apresentados. Este valor

indicará a ordem da função objetivo de cada RNA. Uma vez realizada a consulta das RNAs e já calculada a função objetivo de cada indivíduo, é feita uma ordem descendente (do melhor até o pior) da população segundo seu desempenho. Isto é realizado para selecionar com maior probabilidade aqueles individuos de melhor desempenho.

São aplicados os operadores genéticos sobre cada população local ordenada, gerándose uma nova geração da qual seram escolhidos os de melhor desempenho para substituir os de pior desempenho da geração anterior.

Caso nenhum AG local atinja a convergência, cada AG envia a sua melhor RNA para o Mestre, quem seleciona o melhor de todos e o difunde para todos os nós. Uma vez atingida qualquer convergência local, é aplicada a saída determinada pela RNA obtida como ótima na solução do problema.

5.- CONCLUSÕES E COMENTÁRIOS DO MODELO PARALELO PARA O AGC

A técnica de backpropagation é util nos problemas com treinamento simples. Quando é incrementada a complexidade do problema, o desempenho do backpropagation cai rapidamente. A técnica de busca pelo gradiente tende a chegar e ficar estagnada num mínimo local, do qual o backpropagation com um ganho suficiente pode fugir, mas sem conhecer se o próximo mínimo é melhor ou pior. Dado que o AGC é modelado usando as representações que codificam as soluções mais satisfatórias do problema respeitando a hipótese de armar blocos, os operadores geram melhores filhos dos bons pais, convergendo finalmente para resultados muito perto do ótimo global. Os resultados obtidos com o AGC foram muito bons enquanto a adaptabilidade [SOLAR, 92].

O sucesso de sistemas com meio-ambientes dinámicos sempre requerem de soluções adaptativas. O modelo de AGC tenta resolver os problemas do domínio pela acumulação de conhecimento do problema e usa esta informação para gerar soluções aceitáveis. Estas soluções nem sempre são ótimas, mas geralmente são satisfatórias. Alguns desafios típicos aparecem nas áreas de configuração de sistemas complexos, tais como alocação de tarefas (algoritmo de escalonamento), seleção de rota (problema do caixeiro viajante), e controle de processos. O AGC é uma solução adaptativa que direciona este desafio a modificar a base de conhecimento dinamicamente em resposta às variações do processo [GOLDBERG, 89].

Uma vantagem do modelo de AGC apresentado é a sua característica de operar com um tamanho fixo de população, que reduz a quantidade de memória necessária na implementação. O AGC opera sobre uma grande população de vários grupos de indivíduos distribuidos. A implementação

paralela do AGC resolve o compromiso entre busca genética e tempo real.

Na medida que as mudanças ocorrem, as RNAs são constantemente alteradas, as transformações são continuamente disparadas, e são formados os novos esquemas, dando como resultado uma RNA de "auto-aprendizado".

O AGC têm duas características inherentes que o faz altamente apropriado para o aprendizado por descoberta: mantem uma variabilidade suficientemente alta para prevenir a convergência para ótimos locais; e a categorização e recombinação produz um alto paralelismo implícito.

Da implementação na Rede de Transputers e no hipercubo iPSC da INTEL do modelo °AGC pode-se dizer que verifica que o paralelismo inherente é totalmente explorado em máquinas com baixa granularidade, mostrado pela independencia do tempo com o tamanho da população [pela falta de espaço não é possível inserir as curvas obtidas em SOLAR (92)].

Os testes feitos com problemas difíceis para o aprendizado, tais como OU exclusivo com múltiplas entradas, deteção de inversão e simetría deram excelentes resultados, mostrando a viabilidade do uso de técnicas de computação paralela nos AGs para o aprendizado em RNAs.

Uma pesquisa futura interessante de efetuar é propor uma metodologia de desenvolvimento de AGC para sistemas de aprendizado em tempo real. Isto permitiria desenvolver uma ferramenta de Meta-Algoritmos Genéticos, onde o usuário pode definir as entradas do meio ambiente (detectores), as saídas para o meio ambiente (efetuadores), e eventualmente a função objetivo para determinar o desempenho do sistema no meio ambiente.

REFERÊNCIAS BIBLIOGRÁFICAS

BETHKE, A. (81), Genetic Algorithms as Function Optimizers, *Ph.D. Dissertation*, Dept. of Computer Science, Vanderbilt Univ., Nashville, TN.

BOOKER, L. (87), Improving Search in Genetic Algorithms, in: L. DAVIS (Ed.), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann, Los Altos, CA, 61-73.

BOOKER, L.; D. GOLDBERG & J. HOLLAND (89), Classifier Systems and Genetic Algorithms, *Artificial Intelligence*, Vol. 40, Sept., 235-282.

DAVIS, L. (87), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann, Los Altos, CA.

DeJONG, K. (80), Adaptative System Design: A Genetic Approach, *IEEE Trans. on Systems, Man and Cybernetics*, Vol. SMC-10, Num. 9, Sept., 566-574.

DINNING, A. (89), A Survey of Synchronization Methods for Parallel Computers, *IEEE Computer*, July, 66-76.

FOX, G. (88), *THE THIRD CONFERENCE ON Hypercube Concurrent Computers and Applications, Vol. 1 & 2. Architecture, Software, Computer Systems and General Issues*, ACM, California Inst. of Technology.

GOLDBERG, D. (85), Optimal Initial Population Size for Binary-Coded Genetic Algorithms, *Technical Report TCGA No. 85001*, The Clearinghouse for Genetic Algorithms, Univ. of Alabama.

GOLDBERG, D. (89), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley.

GREFENSTETTE, J. (86), Optimization of Control Parameters for Genetic Algorithms, *IEEE Trans. on Systems, Man and Cybernetics*, Vol. SMC-16, Num. 1, Jan.-Feb., 122-128.

HECHT-NIELSEN, R. (87), Counterpropagation Networks, *Proc. of the Int. Conf. on Neural Networks*, II, 19-32, IEEE Press, New York, June.

HECHT-NIELSEN, R. (90), *Neurocomputing*, Addison-Wesley.

HILLIS, W. (86), *The Connection Machine*, Cambridge, MA, MIT Press.

HINTON, G. (89), Connectionist Learning Procedures, *Artificial Intelligence*, Vol. 40, Sept., 185-234.

HOLLAND, J. (86), Escaping Brittleness: The Possibilities of General Purpose Learning Algorithms applied to Parallel Ruled-Based Systems, in: R. MICHALSKI et alii (Eds), *Machine Learning: An Artificial Intelligence Approach. Vol. II*, Morgan Kaufmann, Los Altos, CA, 593-623.

HOPFIELD, J. (82), Neural Networks and Physical Systems with Emergent Collective Computational Abilities, *Proc. of the Nat. Ac. of Sc. USA*, Vol. 79, 2554-2558.

INTEL CORPORATION (88) *iPSC/2 User's Guide*, Intel Scientific Computers.

MACHADO, R.; C. FERLIN & A. ROCHA (92), Combining Semantic and Neural Networks in Expert Systems, *Relatório Técnico CCR-140 IBM Centro Científico Rio*, Rio de Janeiro, Brasil.

MICHALSKI, R.; J. CARBONELL & T. MITCHELL (86), *Machine Learning: An Artificial Intelligence Approach. Vol. II*, Morgan Kaufmann, Los Altos, CA.

MONTANA, D. & L. DAVIS (89), Training Feedforward Neural Networks using Genetic Algorithms, *Proc. of IJCAI-89*, Detroit, 762-767.

MUHLENBEIN, H. (90), Limitation of Multi-Layer Perceptrons Networks-Step towards Genetic Neural Networks, *Parallel Computing* 14, 249-260.

OOSTHUIZEN, G.D. (87), SUPERGRAN: A Connectionist Approach to Learning, Integrating Genetic Algorithms and Graph Induction, Dept. of Computer Science, Univ. of Strathclyde, Scotland, 132-139.

PALAGI, P. (91), Algoritmos Genéticos no Aprendizado de Redes Neuronais, *Tese M.Sc. do Instituto de Ciências Exatas, Universidade de Brasilia*, Julho, Brasilia-DF, Brasil.

PETTEY, C.; M. LEUZE & J. GREFENSTETTE (87), A Parallel Genetic Algorithm, *Vanderbilt Univ.*, Nashville, TN 37235, 155-161.

PAO, Y. (89), *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Reading, MA.

RIOLO, R. (86), CFS-C: A Package of Domain Independent Subroutines for Implementing Classifier Systems in Arbitrary, user-defined Environments, *Logic of Computers Group*, Division of Computer Science and Engineering, Univ. of Michigan, Ann Arbor.

ROBERTSON, G. (87), Parallel Implementation of Genetic Algorithms in a Classifier System, in: L. DAVIS (Ed.), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann, Los Altos, CA, 129-140.

ROBERTSON, G. (88), Population Size in Classifier Systems, in J. LAIRD (Ed.), *Proc. of the Fifth Int. Conf. on Machine Learning*, Univ. of Michigan, Ann Arbor, 142-152.

SOLAR, M. (92), Modelos Paralelos para o Aprendizado em Redes Neuronais usando Algoritmos Genéticos, *Tese D.Sc. da COPPE/UFRJ*, a ser defendida em 1992, Rio de Janeiro-RJ, Brasil.

SOUCEK, B. & M. SOUCEK (88), *Neural and Massively Parallel Computers. The Sixth Generation*, Fleet Pub.

WHITLEY, D.; T. STARKWEATHER & C. BOGART (90), Genetic Algorithms and Neural Networks, *Parallel Computing 14*, 347-361.

Um Sistema Especialista para auxiliar a Gerência de Falhas em Redes de Computadores

Elizabeth Specialski
Leandro Komosinski
Marcello Thiry

Depto de Ciências Estatísticas e da Computação
Universidade Federal de Santa Catarina
Caixa Postal 476
CEP 88040-900
Florianópolis - SC / Brasil

E-mail: cec1ess@brufsc.bitnet
Fax: 55 (482) 319770

Resumo

Uma das áreas funcionais apresentadas na arquitetura OSI para a Gerência de Redes é a Gerência de Falhas. Dentre as várias atividades existentes neste tipo de gerenciamento, uma diz respeito ao diagnóstico de falhas dos recursos da rede. Este trabalho descreve como técnicas de Inteligência Artificial, em particular os Sistemas Especialistas, podem ser utilizadas para a automatização do processo de geração de diagnósticos e determinação de estratégias para recuperação de falhas.

Abstract

One of the functional areas existing in the OSI Network Management is Fault Management. Among the several activities belonging to this kind of management, one concerns the diagnosis of failures of network resources. This work describes how Artificial Intelligence techiniques, especially Expert Systems, can be used to automatize the process of diagnosis generation, and to determine strategies for failure recovering.

Palavras-Chave: Gerência de Rede, Sistemas Especialistas Gerência de Falhas, Modelo OSI.

Key-Words: Network Management, Expert Systems, Fault Management, OSI Model.

1. Introdução

A diversificação dos serviços oferecidos transformaram as redes de computadores em sistemas complexos, onde o controle e a monitoração tornaram-se necessários para manter o funcionamento global. Esta supervisão é feita através da inclusão de funções de gerência que visam maximizar os recursos disponíveis. Desta forma, pode-se definir o objetivo principal da Gerência de Rede como sendo á manutenção da rede, de modo a garantir sua operação, observando os requisitos de confiabilidade e desempenho pré-estabelecidos. A gerência deve ainda permitir o crescimento ordenado da rede, sem detrimento dos parâmetros de desempenho.

O conjunto de padrões gerenciais propostos pelo modelo OSI [1] pode ser considerado um pacote de ferramentas, as quais podem ser aplicadas ao gerenciamento de diferentes recursos. Os requisitos para a gerência de tais recursos são classificados sob cinco áreas funcionais:

Gerência de Falhas, habilidade de detectar, reportar e coletar estatísticas, além de diagnosticar e corrigir falhas nos recursos;

Gerência de Configuração, habilidade para obter e alterar a configuração de recursos, a fim de manter a operação contínua do sistema;

Gerência de Contabilidade, capacidade de medir o uso dos recursos, identificando os custos envolvidos no consumo destes recursos;

Gerência de Desempenho, capacidade para a monitoração e coleta de informações sobre a operação normal de um sistema, bem como determinar e avaliar o desempenho do sistema sob condições normais ou artificiais; e

Gerência de Segurança, habilidade para controlar o uso de mecanismos de segurança tais como controles de acesso, e a detecção e reportagem de violações de segurança.

Este trabalho atua sob o escopo da Gerência de Falhas (GF), visando a automação dos processos de geração de diagnósticos e correção de falhas, quando estas não necessitarem de intervenção humana imediata. A ocorrência de uma falha em algum recurso da rede pode causar sérios prejuízos para a organização. Este prejuízo tende a crescer quando a falha não é rapidamente detectada ou quando a volta do recurso ao estado operacional apresenta demora excessiva.

Este tipo de problema não apresenta, entretanto, uma solução determinística, sendo necessário a introdução de técnicas de inteligência artificial (IA), em particular os Sistemas Especialistas (SEs), para simular a habilidade que um especialista humano tem de resolver problemas a partir de um conjunto de fatos, que representam a situação momentânea do sistema.

O modelo OSI para a Gerência de Rede, considera os recursos disponíveis na rede como objetos, os quais possuem características próprias que determinam, por exemplo, seu estado operacional. Seguindo esta filosofia, todo conhecimento a respeito do diagnóstico de falhas é modelado na Base de Conhecimento (BC) do SE. Este conhecimento constitui-se basicamente na descrição dos objetos existentes na rede, seus interrelacionamentos e um conjunto de informações heurísticas que conduzem o processo de diagnose.

Após o problema ter sido diagnosticado, o sistema deverá, quando possível, resolvê-lo de forma a seguir a seguinte ordem: a detecção da falha deve ser feita antes de sua ocorrência; caso não seja possível, deve-se procurar contornar os problemas ocorridos tentando manter uma transparência aos usuários; e finalmente se o problema chegar ao conhecimento dos usuários, corrigí-lo rapidamente [2].

O trabalho descreve os problemas encontrados pela Gerência de Falhas e como, através de um exemplo, os Sistemas Especialistas podem ser utilizados na detecção e retificação de falhas.

A seção 2 discute os aspectos relevantes que devem ser considerados em uma Gerência de Falhas. A seção 3 apresenta uma metodologia para a implementação da GF em um ambiente OSI. As seções 4 e 5 descrevem, respectivamente, um sistema especialista baseado em regras e frames e como estes formalismos podem ser utilizados para auxiliar a GF.

2. Gerência de Falhas

Um sistema de gerência de falhas deve ser capaz de identificar comportamentos anômalos dos recursos existentes em uma rede, bem como isolar e corrigir operações anormais do ambiente. Para isto, deve apresentar facilidades para realizar seqüências de testes sobre os objetos gerenciáveis, identificar, diagnosticar e corrigir as falhas ocorridas.

A atividade de gerência de falhas pressupõe o conhecimento prévio das formas normais de operação de cada objeto gerenciável e também dos mecanismos existentes para medir o grau de operacionalidade do recurso. Este é um dos problemas mais antigos na gerência de redes, pois ele trata diretamente com um conjunto de recursos heterogêneos que devem ser monitorados e controlados.

Monitorar é a atividade de recuperar informações referentes ao comportamento de um sistema e controlar é a atividade de trocar o modo como o sistema se comporta.

A questão principal da monitoração e controle é estabelecer uma arquitetura que permita acesso ao recurso de forma independente do equipamento. A melhor solução é aquela usada por todas as arquiteturas de sistemas abertos onde o protocolo de gerência situa-se na camada 7 e manipula uma base de informações de gerência (MIB). O modelo OSI é o melhor exemplo desta solução.

Para permitir a gerência de recursos que não apresentam toda funcionalidade OSI, o modelo proposto considera três tipos de gerência: gerência de sistemas (onde existe a funcionalidade das sete camadas), gerência de camada (que utiliza protocolos de gerência de propósito específico) e a operação de camada (que gerencia uma única instância de comunicação em uma camada).

A MIB consiste em uma base de informações que podem ser transferidas ou modificadas pelo uso dos protocolos de gerência OSI.

Apesar de toda a funcionalidade oferecida pelo modelo de gerência OSI, a gerência de falhas ainda esbarra num problema crucial: como diagnosticar a causa da falha e que ações devem ser tomadas para corrigir a falha? A solução deste problema pressupõe a existência de um especialista e a automatização de suas atividades nos leva à implantação de um sistema especialista para gerência de falhas.

Considerando o trabalho desenvolvido em [3] vislumbrou-se a possibilidade de realizar um mapeamento das facilidades oferecidas pelo modelo de gerência OSI nas facilidades oferecidas pelo mesmo.

3. Metodologia

O primeiro passo para a implantação de um sistema de gerência de falhas é a identificação dos recursos que devem ser gerenciados. A seguir, estes recursos devem ser definidos na forma de objetos gerenciáveis associando-se a eles informações sobre suas características, seu comportamento e as operações que podem ser efetuadas sobre eles.

Considerando-se a existência de uma infraestrutura de gerência OSI implementada, o sistema de gerência apresenta funções que permitem:

- gerenciar objetos (criar, eliminar, trocar valores de atributos)
- relatar eventos (informar e manter histórico de eventos ocorridos)
- descrever a inter-relação entre objetos (estabelecer os relacionamentos entre objetos gerenciáveis)
- relatar alarmes (emitir e manter registro dos alarmes)
- gerenciar testes (estabelecer e realizar testes sobre objetos gerenciáveis)
- monitorar a carga do sistema (observar e medir a carga existente no sistema).

Estas funções auxiliam diretamente a gerência de falhas mas não esgotam toda a sua funcionalidade.

A funcionalidade adicional requerida pelo sistema é encontrada na definição de um conjunto de regras que indicarão as ações mais apropriadas para a resolução dos problemas ocorridos.

O transporte das informações, de notificação e operação é realizado de acordo com o protocolo CMIP (Common Management Information Protocol) [4][5].

4. O Sistema Especialista

Um dos aspectos mais importantes na construção de um Sistema Especialista (SE) é a etapa de modelagem do conhecimento [6]. Para tanto utilizam-se os formalismos clássicos de Representação do Conhecimento (RC) tais como Regras de Produção (RP), Frames, Redes Semânticas, Lógica, etc. Na abordagem utilizada neste trabalho, o conhecimento sobre os objetos a serem gerenciados é modelado através de Frames e RP. Estes dois formalismos, descritos abaixo, fazem parte do sistema ADSBC [3] utilizado na implementação.

O formalismo de Frames utilizado baseia-se nos conceitos de classes de objetos, instâncias e herança de características. Tais conceitos são idênticos àqueles existentes no paradigma de Orientação à Objetos. O que torna os Frames diferentes dos objetos são os princípios que regem seu comportamento. Nos objetos os procedimentos (chamados métodos), que estão associados ao objeto, são executados e, como efeito, modificam/recuperam os valores dos atributos. Já nos Frames, os procedimentos (chamados de demons), que estão associados aos slots (equivalentes aos atributos) são executados como efeito colateral ao fato do sistema querer acessar/modificar os valores dos slots.

Outra característica exclusiva dos Frames é que em cada slot existe uma definição que determina quais são os valores válidos (consistentes) para o slot. Deste modo, cada instância sempre se mantém consistente e qualquer tentativa de provocar uma inconsistência será automaticamente rejeitada. Além disso, através do mecanismo de demons pode-se associar um procedimento que realizará as ações pertinentes quando da tentativa de atribuir um valor inconsistente a um slot. Por exemplo, o Frame pode emitir uma mensagem para o usuário ou então emitir uma notificação qualquer.

As Regras de Produção (RP) no sistema ADSBC podem ser usadas com dois objetivos: determinar o valor de um slot e realizar o processo principal de inferência. No Sistema Especialista que está sendo desenvolvido, os Frames são usados para modelar os objetos gerenciáveis definidos pelo padrão OSI. As RP atuam como mecanismo principal de inferência e são responsáveis pela emissão de diagnóstico bem como sugestão de ações que devem ser tomadas para recuperar uma situação de erro.

5. A Utilização do Sistema Especialista na Gerência de Falhas

No sistema proposto a modelagem dos objetos gerenciáveis de uma Rede obedece o seguinte critério: cada classe de ob-

jetos gerenciáveis (como por exemplo impressoras, CPUs, canais físicos, etc.) é modelada através de um Frame. Cada instância de uma classe de objeto gerenciável é modelada por uma instância de Frame.

No contexto de Gerência de Falhas, os atributos que estão definidos para cada Frame justificam-se pela necessidade de se conhecer tal informação para gerar um diagnóstico e/ou executar as ações de tratamento das falhas. Isto significa que a medida que outros serviços de gerência forem sendo definidos, novos atributos serão definidos para cada Frame.

Modelagem da Classe Impressora

Neste item são apresentados, como exemplo, os aspectos relevantes à Gerência de Falhas para a modelagem da Classe Impressora. Os atributos são divididos em grupos:

Grupo 1 : Identificação da Instância

Neste grupo os atributos identificam unicamente a instância.

* identificação (um código. Por exemplo : i1, i2, i3, ...)
* classe = impressora

Grupo 2 : Informações Gerais da Instância

Neste grupo os atributos contém informações gerais que, embora caracterizem o estado da instância, não têm nenhuma influência sobre os possíveis problemas de falha.

* lista dos programas na fila de impressão

Grupo 3 : Informações para Gerência da Instância

Neste grupo estão os atributos que são utilizados pelo sistema de Regras de Produção para gerar diagnósticos e definir ações para recuperar o erro.

* tipo (matricial / laser / etc.)
* qualidade de impressão (rascunho / normal / artigo)
* tamanho máximo da fila de impressão
* tamanho atual da fila de impressão
* estado operacional (habilitado / desabilitado)
* estado de uso (ativo / ocupado / ocioso / desconhecido)
* estado administrativo (bloqueado / não bloqueado)
* estado de gerência (fazer manutenção / ok)
* backup (sim / não)
* período máximo sem manutenção preventiva
* tipo da falha (papel / fita / comunicação / etc.)
* tempo em que a impressora está com falha
* tempo máximo que a impressora pode ficar com falha
* grau de importância associado à instância (pouco / médio alto)

Observando os atributos do grupo 3, percebe-se que a estratégia a ser utilizada na Gerência de Falhas permite ao Sistema Especialista atuar em duas situações distintas. Na primeira, o SE pode gerar diagnósticos e propor ações corretivas após a falha ter ocorrido. Na segunda, o SE pode prever uma falha eminente, agindo antes do problema ocorrer de fato.

Para o caso específico da classe Impressora, os diagnósticos são gerados por uma palavra de estado que indica a situação em que a mesma se encontra. Assim, o mais importante é determinar que ação deve ser tomada para recuperar o erro.

O sistema de gerência, para tomar decisões a partir dos fatos obtidos nos diagnósticos, necessita de um formalismo que modele as ações a serem executadas. Deste modo, associadas aos frames, as regras de produção determinarão a conduta do sistema.

A seguir está descrito um conjunto de regras que expressa como pode ser mapeado o conhecimento de algumas situações.

Regra R1:
Se a impressora I apresenta falha e
existe uma impressora I1 de reserva e
o grau de importância da impressora é alto
Então
passar todo o estado da impressora I para a impressora I1

Regra R2:
Se a impressora I apresenta falha e
não existe outra impressora de reserva e
o grau de importância da impressora é alto ou médio e
existe outra impressora I1 com as mesmas características de I e
o tamanho da fila de impressão de I é TamFI_I e
o tamanho da fila de impressão de I1 é TamFI_I1 e
(TamFI_I + TamFI_I1) é menor que o tamanho máximo da fila de I1
Então
passar todo o estado da impressora I para a impressora I1

Regra R3:
Se o período máximo sem manutenção da impressora I é P e
o valor de P está próximo de um limite (thresold)
Então
avisar que a impressora I deverá ser substituída para que se possa fazer uma manutenção preventiva

Regra R4:
Se a impressora I deve sofrer manutenção e
o estado operacional de I é habilitado e
o estado de uso é ocioso e
Então
mude o estado operacional de I para desabilitado

Regra R5:
Se a impressora I apresenta falha e
I tem tempo máximo de falha TMax_Falha e
I tem tempo de falha T_Falha e
T_Falha > TMax_Falha
Então providenciar recuperação da impressora I

6. Conclusão

A utilização de um SE como auxílio no processo de Gerência de Falhas possibilita uma agilização dos procedimentos de tomada de decisão na resolução de problemas. Esta agilização é alcançada, principalmente, pela automatização da geração de diagnósticos e das ações realizadas na recuperação do sistema aps a ocorrência de falhas.

Outro aspecto que influencia o sucesso de um sistema para GF é a possibilidade de prever problemas, diagnosticando-se com antecedência que o estado atual do sistema pode levar à ocorrência de falhas. Pode-se então executar ações que eliminem a falha antes dela ocorrer. Desta forma, assegura-se a estabilidade da rede, ou seja, sua operação contínua.

7. Referências Bibliográficas

[1] ISO/IEC 7498-4, Information Processing Systems, Open Systems Interconnection, Basic Reference Model Part 4 - "Management Framework", First edition, 1989.

[2] Ball, Larry L., "Cost-Efficient Network Management", Mc-Graw-Hill Series on Computer Communications, 1992.

[3] MUELLER, A., CUNHA, F. & KOMOSINSKI, L. J., "Um Ambiente de Resolução de Problemas para Sistemas Baseados em Conhecimento", anais da XVIII Conferência Latinoamericana de Informática, Ilhas Gran Canárias, 1992.

[4] ISO/IEC 9595, Information Technology, Open Systems Interconnection, Common Management Information Service Specification, 1989.

[5] ISO/IEC 9596, Information Technology, Open Systems Interconnection, Common Management Information Protocol Specification, 1990.

[6] KOMOSINSKI, L. J., "Uma Linguagem Centrada em Frames para Desenvolvimento de Sistemas Baseados em Conhecimento", dissertação de mestrado, Universidade Federal de Santa Catarina (Brasil), março, 1990.

DEVELOPING DISTRIBUTED APPLICATIONS UNDER THE OBJECT ORIENTED MODEL

Eleri Cardozo
Tech. Institute of Aeronautics
12225 Sao Jose dos Campos - SP

Tadao Takahashi
Ethos Project
13083 Campinas - SP

*** DRAFT ***

1. INTRODUCTION

Object oriented programming (OOP) [1,2] is becoming a standard design methodology for large software system developments. OOP favors a programming style where the software components (objects) are nearly self-contained (many of them constructed by specializing existing ones), have well defined roles and clear mechanisms for interacting with other objects. Such properties decrease the efforts demanded by the implementation of complex software systems and increase the system's reliability, testability and maintainability.

Unfortunately, OOP techniques are available today at the programming language level and few extensions of such techniques to distributed environments have been reported [3]. This paper proposes an extension of DPSK (a Distributed Problem Solving Kernel) [4] in order to accommodate some OOP practices like hierarchical organization of classes, instantiation of classes, definition of methods, etc.

The main objective of this research is to derive an environment for distributed problem solving (named DPSK+P) that has the advantages of DPSK in terms of convenience and efficiency, plus some added capabilities that favor object oriented programming at the distributed application level.

2. OBJECT-ORIENTED PROGRAMMING

The object oriented programming approach is relied on two concepts: objects aggregated into classes, and methods. An object can be defined as a black-box modeling some entity of the domain. This black-box packs the data necessary to quantify the entity plus, a set of procedures that manipulate the data. Objects interact by defining procedures that can be activated by other objects. Such procedures are called methods. A method can have input parameters, and once activated produces side effects and may return some useful result. The object's internal data has a public portion and a private portion. Public data can be accessed from other objects within the same program, while private data are manipulated exclusively by the object's internal procedures. Figure 1 schematizes an object. Since objects are nearly self-contained, a better structuring of the entire code is achieved. As a result, productivity, extensibility, testability and reliability are favored by OOP when compared to other programming techniques.

A central concept in OOP is the concept of class. A class defines a behavior for a family of objects derived from it (the class' instances). A class keeps the object's data and methods[1], which are inherited by its instances. Some classes derive from one or more classes, forming an hierarchy of superclasses (ancestors) and subclasses (descendants). A class inherits the behavior (data and methods) from its direct superclasses, elaborating this behavior by

1 Some object oriented systems keep methods separated from classes (CLOS [8] is the best example of such systems).

specializing the data and by adding new features to the methods.

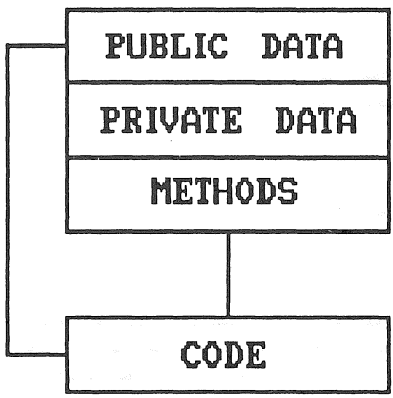


Fig 1. An object and a code (other objects) using it.

A program elaborated according to the object oriented model is composed of a hierarchy of classes and a set of instances (objects) interacting among them. For example, in the design of a drawing package, one can define a class named "shape". Geometric figures like circles and rectangles are specialization of shapes, squares are specializations of rectangles, and so on. Methods like "display" can be defined for the class "shape" and inherited by all the instances belonging to the subclasses of "shape". More specialized methods like "rotate" are defined for each geometric figure, possibly by elaborating those inherited from the respective superclasses. Calling "rotate" for an argument belonging to the class "square" will automatically invoke the method that rotates squares. This property is called polymorphism, and the binding between methods and calling parameters can be accomplished at compiling time (early binding) or at run time (late binding).

3. DISTRIBUTED PROBLEM SOLVING

In short, Distributed Problem Solving (DPS) is a methodology for software developments that favors a set of autonomous modules (the problem solving agents) running independently in a multitasking or multiprocessing machine or in a set of processors linked through a local area network [5].

The modules need to cooperate in order to achieve a common goal: the solution of the problem being processed. Cooperation is composed of two basic activities: communication and control. Communication is the exchanging of information among the modules and control is the coordination necessary to guarantee that each module will perform its task at the right time and with the proper resources allocated to it. Basic control activities are inter-module sequencing (managing tasks like the access to critical resources) and execution control (managing tasks like starting of modules).

Inter-agent communication can be achieved through mechanisms based on message passing, sharing of data, remote procedure call, etc. Mechanisms for control include semaphores, events, barriers, etc (for sequencing) added to the proper operating system directives for execution control. Several tools for aiding DPS developments have being designed and implemented. Such tools provide communication and control services based on one or more mechanisms mentioned above.

The basic communication activities found in centralized object oriented programming languages and systems can be extended to distributed environments. Communication in OOP is based on the calling of methods and sharing of data (the public portion of the object's data). In distributed environments, the calling of methods may be relied on a client/server communication protocol (like rendezvous or remote procedure call); while the sharing of data needs only an accessing discipline to keep the data consistent and correct under concurrent access.

The model for distributed computing proposed in this paper is based on the concept of shared objects. A shared object is a macro structure aggregating an amount of public (shared) data and a set of active methods[2]: the smallest amount of computation managed by the operating system (processes in the case of UNIX). Active methods can be a set of ordinary objects if coded according to the OOP model, or any piece of sequential code. Without loss of generality, we assume that each problem solving agent in a distributed environment is an active method attached to some shared object.

Shared objects interact among them by sharing their public data added to a client/server communication protocol for accessing the active methods. The public data is a set of attribute-value pairs (slots), and the protocol for method activation remembers the rendezvous mechanism found in the ADA programming language. Figure 2 shows schematically a shared object.

To illustrate the usage of shared objects as an integration medium, consider an environment for system's identification, optimization and control as described in [6]. This environment has several modules (integrated around an executive) performing optimization, real time control, system's identification, etc. If these modules are to be distributed through many processors, these services could be understood as shared objects performing well defined tasks. Taking system's identification as an example, algorithms as Recursive Least Squares, Extended Least Squares, etc, are active methods of the shared object performing system's identification. Separated active methods can be employed for each algorithm, or more than one algorithms be grouped into an active method. In the last case, the appropriate action (algorithm) is selected according to the parameters passed to the active method during its invocation (see section 7).

2 All the code attached to a shared object will be referred as a method, even those not performing interfacing with other shared objects.

Even if the system's modules are not coded according to the object oriented model, the concept of shared object brings to the system's architecture the central ideas behind object oriented programming. As a result, it is expected an increasing in many desired attributes like reliability, testability, etc.

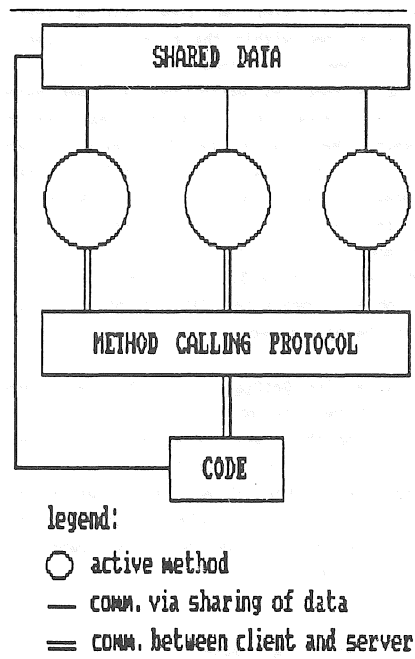


Fig 2. A shared object.

4. COMMUNICATION IN DPSK+P

As stated before, communication in DPSK+P is relied on the sharing of data and calling of active methods. The following subsections describe how shared objects can be employed for communication.

4.1 Defining Classes

DPSK+P allows the user to define a hierarchy of classes shared by all the problem solving components. A class is defined by a set of

slots (attribute-value pairs) and may be derived from another classes forming a graph of superclasses and subclasses. Classes are identified by a global name. A single primitive[3] is supplied for class definition: DefineClass. The primitive is employed for building a class giving its direct superclasses, its data attributes (slots), and its active methods (the respective addresses of compiled code holding them). Slots defined within the direct superclasses are automatically inherited by the class being defined. By definition, a class named "t" is a superclass of every class (except itself). New classes can be created at any time and redefining an already existing class causes the replacement of the old definition by the new one. In such case, all the instances of the old class are marked as "obsolete" (see 4.3). The primitive returns an object identifier (ID) pointing to the class being created.

The primitive GetClass takes the class' name and returns its object ID. Primitives for navigating the class hierarchy are provided: GetParents and GetChildren returns respectively the direct superclasses and subclasses of a given class. Once the class structure has been defined, the agents can start to create instances of classes.

4.2 Instantiating Classes

A shared object is an instance of some class (shared object and instance will be employed as synonymous). Instances of classes are created by calling the MakeInstance primitive. This primitive takes the pointer to the class (returned by DefineClass or GetClass) and a set of slots to be added or to replace those inherited from the class. After the shared data has being built, a special method called "constructor" is invoked (if defined by the class). The constructor initializes the shared data, calling other methods if necessary. MakeInstance returns an object ID pointing to the shared object just created.

If not specified otherwise, shared objects are created in the processor where the calling agent is running (not where the class is defined as in the current version of DPSK). An optional third argument in MakeInstance directs the creation of the instance to a different processor.

4.3 Accessing and Updating Instances

Instances can be accessed by the GetInstances primitive. This primitive accepts as input a pointer to the class and a set of specialization slots that must be present in the instances accessed. A list of object IDs pointing to the instances satisfying the given specialization slots is returned.

With the object ID pointing to an instance it is possible:

a. To access any slot in the instance.
b. To delete, add or update a particular slot in the instance.

Accessing or updating an obsolete instance will cause the calling of a special active method named "repairer" (if defined within the class). This method has the mission of making the instance built under an older version of the class compatible with the new definition of it. The mark "obsolete" is removed when the repairer exits.

4.4 Deleting Instances

The primitive DeleteInstance destroys a shared object. An object ID pointing to a deleted object becomes unbounded, turning without effect all operations over it. Immediately before the instance destruction, a special active method called "destructor" is invoked (if defined in the class). This method performs the clean-up relating the instance under destruction (perhaps calling other methods informing the elimination of the shared object). The object is unbounded after the destructor exits. Active methods attached to the a deleted object are killed.

---

3 The term primitive is used interchangeably to refer a C, C++ or Lisp function; a C++ member function; or a CLOS [8] generic function.

4.5 Defining and Calling Shared Methods

Until now we noticed the existence of three special methods supplied by the user when defining a class: the constructor, destructor and repairer of instances. Any other active method can be defined by given them different names. Active methods defined for a class are instantiated (started) in behalf of a shared object when a CallMethod or StartMethod primitive is called given the shared object as target. After its starting, a method can exit after the processing of the call, or remains up waiting for new requests. Each method has a code implementing it (again, a process in UNIX). When many lightweight methods are defined for a class, all of them can be grouped into a dispatcher. The dispatcher selects the desired method by the input parameter in the call (see section 7).

A method is invoked by calling the CallMethod primitive. This primitive takes three parameters: the target--a class or instance to which the call is directed, a single object encoding the input parameter of the call (that in its turn can hold many other objects), and a timeout for the calling request. The caller remains blocked until the call is processed by an active method or a timeout occurs.

The protocol for calling methods is as follows. Calls can be directed to classes or instances. Calls directed to instances can be processed only by active methods attached to this instance. Calls directed to classes can be processed by active methods attached to any instance of the class. When a call to the active method M is directed to an instance:

a. If M is already running within the instance, then queue the call; otherwise

b. If M is defined in the instance's class, then start a clone of M within the instance and queue the call. If the class does not define such method

c. Move up in the class hierarchy until M is found. Start a clone of M within the instance and queue the call. If the method can not be found, return a null value for the call.

Notice that item c. provides inheritance of methods.

If a call is directed to a class, it is queued in the class until an instance accepts it (no method is started).

Active methods accepting calls are servers that keep performing the following cycle :

a. Get a call request.

b. Process the call.

c. Return the result to the caller.

d. Exit or go to a.

Requests are accepted by active methods through the primitive AcceptCall. This primitive has a timeout which controls the amount of time the active method wishes to wait for a call. The method becomes blocked until the arriving of a call for it or a timeout occurs. The primitive returns the parameter passed by the other end (the caller) and an object identifying the Call-Accept pair. When the method finishes the processing of a call, the primitive ReturnCall completes the communication by returning the result of the computation to the caller. The accepting protocol is as follows:

a. If there are pending calls for this method directed to this shared object, process the oldest call, otherwise

b. If there are pending calls for this method directed to the class, process the oldest call.

If no call is pending to the accepting target, steps a. and b. are performed until the timeout specified in AcceptCall occurs.

Active methods are selected by name and by the object holding the method (as in the Flavor System [7]). No method combination

strategy is provided by DPSK+P. Binding occur at run time.

Active methods can discover the object they belong by calling the primitive GetWorld. The primitive returns a pointer (object ID) to the shared object the method belongs. With this pointer the method can operate in the object's shared data.

4.6 Remarks on the DPSK+P Communication Structure.

The communication structure adopted in DPSK+P attends both data-driven and goal- driven problem solving strategies. Data-driven strategies will use the manipulation of shared data to progress the problem solving activity while goal-driven strategies will use the method calling protocol in order to invoke methods for solving subproblems generated during the planning/decomposition phase.

With these two complementary communication structures, we believe that DPSK+P fulfills the communication requirements for a wide range of practical applications.

5. CONTROL IN DPSK+P

5.1 Locking

Under concurrent access, the shared objects may become inconsistent if proper control is not provided. The accessing of the shared data kept by the shared object needs a locking mechanism that assures consistency and correctness of the shared objects. DPSK+P provides a locking mechanism based on transactions [8].

Transactions can be performed over shared objects, classes or hierarchy of classes. The scope of the locking is the following:

a. A transaction over a shared object will lock only the shared data kept by the object.

b. A transaction over a class is equivalent to a transaction over all the shared objects belonging to this class.

c. A transaction over a hierarchy of classes is equivalent to a transaction over the starting class and all of its subclasses.

The scope of the locking ranges from the shared data kept by a single shared object to the entire shared data (transaction of a hierarchy of classes starting at the class "t").

A shared object is not available for locking if:

a. Its constructor is still running, or

b. Its destructor is already running, or

c. The object is already locked for read/write, or

d. The object is already locked for read-only and a read/write lock is being requested.

A lock is obtained by calling the BeginTransaction primitive. This primitive accepts an indicator of locking type (read/write or read-only) and a list of objects to be locked. The primitive returns a locking object if the transaction was accepted, or a null object otherwise. The primitive EndTransaction and AbortTransaction finishes and aborts a transaction in progress. Both primitives accept the locking object returned by BeginTransaction as argument.

If serializability only is demanded (without atomicity and persistence), a simple and faster mechanism based on the method calling protocol can be implemented (as illustrated in the section 7).

5.2 Execution Control

In the object oriented model data and code are aggregated into objects. Following this line, DPSK+P allows the control of the code running within the active objects, that is, the shared methods. Two primitives are

provided: StartMethod and InterruptMethod. StartMethod starts an active method attaching it to some given shared object (duplication of an active method in the same object is allowed). The method must be defined in the object's class or in any of its superclasses. InterruptMethod sends a signal to an active method of a given shared object. Signals provided are those found on UNIX like:

a. KILL: kills the method.

b. SUSPEND: freezes the execution of a method.

c. RESUME: resumes the execution of a suspended method.

d. HANGUP: hangs-up (branches the execution to a given exception handler) a method.

e. Any other signal.

The handling of the signal is user's responsibility (and not possible in some programming languages).

5.3 Synchronization

The basic synchronization mechanism provided by DPSK+P is the method calling protocol. Using this protocol as a basis, it is simple to implement other synchronization mechanisms like events, semaphores, barriers, etc. Section 7 illustrates this feature.

6. INTERFACES

DPSK+P will provide interfaces for C, C++ and CLOS (Common Lisp Object System) [9] programming languages. The C interface is based on a library of functions, while the C++ and CLOS interfaces are based on a set of classes and some few functions. The C++ interface to DPSK+P is listed in the appendix A.

DPSK+P itself is implemented in plain C to favor transportability to more specialized hardware systems. For inter-machine data transferring, the UNIX interface to the TCP/IP protocol is employed. Data coding/decoding follows three different protocols. When the network is homogeneous, the binary protocol is the fastest available since data structures are shipped in their in-memory format (without coding). For heterogeneous networks two protocols are available: the eXternal Data Representation (XDR) [10] protocol (when available in all machines), or a protocol based on the ASCII code (the most general and slower available).

The kernel itself can be replicated in order to avoid interference between different applications when running at the same time.

7. AN EXAMPLE

This section shows how a synchronization mechanism based on semaphores [11] can be implemented using the method calling protocol.

The semaphore mechanism relies on two functions: P and V. When the process desires to enter the critical region it calls P. Upon returning of P it executes the critical section, calling V at the end. P has the following structure:

a. Subtract 1 from the value of the semaphore.

b. If the value is non negative return, otherwise

   i. Add the caller process to the list of blocked processes.

   ii. Block the process.

V returns if the semaphore has its value grater than the unit, blocking the caller if the value is zero or less.

The V function has the following structure:

a. Add 1 to the value of the semaphore.

b. if the value is less than or equal to zero:

   i. Remove one entry from the blocked processes.

ii. Wake up this process.

V makes a blocked process on a P call to return, granting it access to the critical region.

An interface to the semaphore mechanism can be implemented as a C++ class that defines, operates and destroys semaphores. The member functions of this class access DPSK+P in order to achieve the proper synchronization among the distributed system components.

To implement the semaphore mechanism, one can define a DPSK+P class named "semaphore" with a single active method called "PV" that implements both functions P and V. The class has a single slot ("name"). The value of the semaphore will be kept by the method PV. The class semaphore will have an instance for each semaphore defined.

The interfaces for P and V will call the method "PV" with an object as input parameter stating if the call is to be processed as "P" or "V". The method "PV" will keep a list of blocked process (step b.i of V). Callers will be blocked by "PV" simply by not returning the call. In step b.ii of V, the caller will be waked-up by returning the respective call. The list of blocked processes is, in fact, a list containing Call-Accept identifiers (see 4.5).

The code of this example is presented in the Appendix B.

8. CONCLUDING REMARKS

DPSK+P has some attractive features not found in the current version of DPSK. The communication structure found in DPSK was improved by adding a hierarchical organization of classes with inheritance of slots and methods. The method calling protocol allows a client/server communication structure (suitable for goal-driven problem solving strategies) plus a wide range of synchronization mechanisms to fit the needs of practical applications. Libraries of classes (like the class "semaphore" defined above) can be incrementally build and added to the kernel. A monitoring facility is also under development. Crash recovery and migration of shared objects started to be investigated.

9. ACKNOWLEDGEMENTS

This research is being supported by the National Science Foundation, Grant Number CDR-8742796 at the Engineering Design Research Center, Carnegie Mellon University; and Ethos Project at the Tech. Institute of Aeronautics, Brazil.

10. REFERENCES

1. Cox, B., Object Oriented Programming: an Evolutionary Approach, Addison-Wesley, 1987.

2. Meyer, B., Object-Oriented Software Construction, Prentice-Hall International, 1988.

3. Larner, D.L., Factories, Objects & Blackboards, AI Expert, April 1990.

4. Talukdar, S. and Cardozo, E., Building Large Scale Software Organizations, in Expert Systems for Engineering Design, M.D. Rychener (editor), Academic Press, 1988.

5. Demazeau, Y and Muller, J.P. (editors), Descentralized A.I., Elsevier Science Publishers B.V., 1990.

6. Hemerly. N.M., PC-Based Packeges for Identification, Optimization and Adaptive Control, IEEE Control Systems Magazine, Vol. 11, No. 2, Feb 1991.

7. Sun Microsystems Inc, Sun Common Lisp 3.0 - The Flavor System, First Edition, August 1988.

8. Date, C., An Introduction to Database Systems, Vol II, Addison-Wesley, 1983.

9. Keene, S.E., Object-Oriented Programming in Common Lisp, Addison-Wesley, 1989.

10. Sun Microsystems Inc, Network Programming, Part Number 800-1779-10, May 1988.

11. Peterson, J.L. and Silberschatz, A., Operating Systems Concepts, Second Edition, Addison-Wesley 1985.

An M.Sc. Level Course of Studies in Parallelism

Alejandro Teruel[1], Marianela Lentini[2], Carlos Figueira[1]
Universidad Simón Bolívar, Apartado 89000
Caracas 1080A, VENEZUELA

Abstract
Two years ago, at the Universidad Simón Bolívar, we designed and successfully implemented an ongoing set of M.Sc. level courses in Parallelism to help overcome the problems besetting graduate students who wish to specialize in the area and tackle problems such as choosing a parallel programming hardware and software platform, developing parallel programs for it and building the necessary tools to support parallel software development.

The purpose of this paper is to present the current status of the Universidad Simón Bolívar's course of studies in Parallelism.

Keywords: Parallelism, education, softwaredevelopment.

Introduction

Choosing a parallel programming hardware and software platform, developing parallel programs for it and building the necessary tools to support parallel software development all require a thorough knowledge and understanding of many diverse and interleaved issues in parallel hardware architecture, parallel algorithm design, tools for parallelism and the theory of parallel processes. A formal course of studies in such topics is becoming necessary in order to provide a coherent structure to what otherwise becomes a scattered, haphazard, repetitive, superficial or, at best, lopsided coverage of topics. Two years ago, at the Universidad Simón Bolívar (USB), we designed and successfully implemented an ongoing set of M.Sc. level courses in Computer Science to help overcome such problems. The purpose of this paper is to present the current status of the USB's course of studies in Parallelism.

Objectives

The current course of studies emphasizes the MIMD distributed memory model of parallelism. This emphasis is due mainly to the local demand for our M.Sc. graduates and to the hardware acquisition restrictions imposed by our extremely small budget. However, the course stucture is independent of this bias and its contents can easily be

[1] Department of Computer and Information Technology
[2] Department of Mathematics

focused towards other models.

The objectives of the course of studies are:

- To provide a coherent coverage of the field of parallelism;
- To prepare students for the wide variety of roles they will have to play in the near future as regards the selection, development, and use of parallel technology.

Obtaining the fastest possible running parallel programs is the raison d'etre in the field of Parallelism today, hence the disproportionate intertwining of programming techniques, performance analysis and software and hardware platform interdepencies that must be taken into consideration during parallel software development. While we have found it convenient to factor out topics into the more conventional categories of hardware (Parallel Architectures), system software and tools (Parallel Tools) and application software (Parallel Algorithms, Parallel Numerical Methods), each course can be better described as emphasizing one category's point of view over the others rather than narrowing down on the study of any one category to the exclusion of the others --the latter simply misses the nature of the whole field.

Requirements

The course of studies in parallelism presupposes basic knowledge of computer architecture, concurrency, operating systems, algorithm design and graph theory, as provided in a typical undergraduate curriculum in Computer Science. Students with a major in related fields such as Mathematics and Electronic Engineering have also been able to successfully complete the course of studies.

Courses

The course of studies consists of the following courses:

- Parallel Architectures;
- Parallel Algorithms;
- Parallel Numerical Methods I;
- Parallel Numerical Methods 2;
- Tools for Parallelism;
- Theory of Parallel Processes.

In order to satisfy the M.Sc. degree requirements in Computer Science at the USB, candidates must successfully complete at least eight Computer Science courses and an M.Sc. thesis. The course work must include the following kernel:

- Mathematics for Computer Science;
- Algorithm Analysis;

- Theory of Computation.

The planned duration of studies leading up to the degree is four twelve-week terms; this spans a calendar year and a quarter.

A more detailed description of each course follows.

ParallelArchitectures

The objective of this course is to provide a programmer of parallel machines with the knowledge necessary to:

- Choose the best parallel architecture model for a given problem;
- Understand the limitations of a specific parallel machine or architecture;
- Tune programs taking into account the features of a specific parallel architecture.

Course outline

The course looks at ways of speeding up the key subsystems in order to prevent their becoming bottlenecks on parallel machines. The impact of such architectural decisions on cost, speed, scaleability and tool and programming complexity is also carefully considered. A highly recommended form of drawing all the threads of the course together at its end is setting a final assignment which consists of selecting a parallel machine for a (fictitious) client company operating in a specific industrial sector (e.g. oil exploration and extraction or prothesis design) and using order of magnitude restrictions on the budgeted cost of the system. Ease of operation, use and programming as well as software availability, functionality and performance must also be taken into account, together with more specific hardware and supplier characteristics.

Topics

- Introduction.

  Why parallelism? Parallelism vs. distributed systems. Flynn's taxonomy of parallel architectures. Applications of parallelism.
- Measuring performance on parallel architectures.

  Throughput and latency. Speedup and Amdahl's Law. Peak and sustained performance. MIPS, MFLOPS and workloads. Using kernels and benchmarks.
- Pipelining and Vector machines.

  Pipelines and their hazards. Superscalar machines. Vector registers and vector instructions. Stride. Interleaved memories and memory conflicts. Chaining. Vector masks. Hockney's metrics: n_v, $n_{1/2}$, r_∞. Cache and vector machines.
- MIMD machines.

  Shared memory vs. distributed memory. Interconnection networks for shared

memory machines: Clos networks vs. Banyan networks. Synchronization methods. Cache on multiprocessor systems. Interconnection networks for distributed memory machines: fixed topology vs. reconfigurable topologies. Cut-through routing, wormhole routing. Input/output subsystems: Disk arrays.

- SIMD machines and massive parallelism.
- Other architectures.

  Systolic architectures. Dataflow machines.
- Case studies.

  Varies from year to year. In 1991, case studies included Intel i860 and INMOS transputer platforms, the IBM 3090VF, nCube and Maspar parallel machines.

References

No textbook covers all the required topics with the necessary level of detail. The following set of references complement each other and are listed in roughly descending order of use:

H. Stone: "High-Performance Computer Architecture". 2° Edition, Addison-Wesley, 1990.

B. Wilkinson: "Computer Architecture: Design and Performance". Prentice-Hall, 1991.

D. Patterson, J. Hennessy: "Computer Architecture: A Quantitative Approach". Morgan-Kauffman, 1990.

G. Almasi, A. Gottlieb: "Highly Parallel Computing". Benjamin/Cummings, 1989

K. Hwang, F. Briggs: "Computer Architecture and Parallel Processing". McGraw-Hill, 1984.

D. Bertsekas, J. Tsitsiklis: Parallel and Distributed Computation: Numerical Methods". Prentice-Hall, 1989.

R. Hockney, C. Jesshope: "Parallel Computers 2". Adam Hilger, 1988.

Some topics require more specific treatment. A complete list of references for the course is available from the authors.

Parallel Algorithms

The emphasis of the course is on the design and implementation of efficient parallel algorithms. In general, first algorithms for PRAM models are developed and compared and then their adaptation to actual MIMD machine with distributed memory is studied, attempted and compared to known solutions, where they exist. Programming some of the algorithms is an indispensable part of the course, since it allows the student to grapple with the characteristic difficulties of the area e.g. load balancing, efficient communications and steering clear of deadlocks.

Topics

- Introduction

  The PRAM models (EREW, CREW, CRCW). Distributed memory MIMD

architectures. Measuring performance: speedup, efficiency and cost. A parallel programming language and platform (used for programming assignments e.g. INMOS Parallel C on a transputer based system).

- Parallel searching, sorting and merging.
  Searching. Prefixed sums. Selection. Merging. Sorting.
- Parallel graph algorithms.
  Basic matrix operations. Finding the connected components. All pairs shortest paths. Topological sort. Minimum spanning tree. Eulerian circuits. Travelling salesman and the simulated annealing heuristic.
- The class of inherently sequential algorithms.

The rest of the topics covered are chosen from the following list, time permitting:

- Data structures and parallelism: hashing, heaps, priority queues.
- More graph algorithms: maximal matching, graph colouring, network flows.
- Pattern matching.
- Alpha beta tree search.
- Computational geometry.
- Optimization.

References

The basic textbook for the course is:

S. Akl: "The Design and Analysis of Parallel Algorithms". Prentice-Hall, 1989.

This needs to be complemented by a book which deals with some of the nasty surprises that can crop up for the unwary programmer:

G. Fox , M. Johnson, G. Lyzenga, J. Salmon, D. Walker: "Solving Problems on Concurrent Processors". Volume 1. Prentice-Hall, 1988.

Additional material can be found in:

D. Bertsekas , J. Tsitsiklis: Parallel and Distributed Computation: Numerical Methods". Prentice-Hall, 1989.

S. Bokhari: "Multiprocessing the Sieve of Eratosthenes". Computer, April 1987.

A. Rytter, W. Gibbons: "Efficient Parallel Algorithms". Cambridge University Press, 1988.

Y. Won. S. Sahni: "A Balanced Bin Sort for Hypercube Multicomputers". Journal of Supercomputing, 2 (1988).

Y. Won, S. Sahni: "Hypercube to Host Sorting". Journal of Supercomputing, 3 (1989).

Parallel Numerical Methods 1,2

These two courses provide a thorough grounding in the state of art in the development of numerical algorithms for parallel architectures, with emphasis on MIMD distributed memory machines. A number of alternatives to each problem are considered; each algorithm is analyzed for efficiency, stability and consistency.

Topics

- Review of parallel architecture and other basic concepts in parallelism;
- Common communication requirements of numerical algorithms;
- Parallel implementation of BLAS routines.
- Direct solution of linear equations. Dense and sparse problems.
- Iterative solution of linear and nonlinear systems.
- Unconstrained optimization. Decomposition of optimization problems.
- Asynchronous iterative algorithms.

References

The main references for the course are:

D. Bertsekas, J. Tsitsiklis: Parallel and Distributed Computation: Numerical Methods". Prentice-Hall, 1989.

J. Modi: "Parallel Algorithms and Matrix Computation". Clarendon Press, 1988.

M. Lentini: "Introducción a los Algoritmos Numéricos en Paralelo". Segunda Escuela Venezolana de Matemáticas, 1989.

M. Ortega: "Introduction to Parallel and Vector Solution of Linear Systems". Plenum, 1988.

Additional material includes:

Y. Saad, M. Schultz: "Data Communication in Parallel Architectures". Yale University Research Report. YALEU/DCS/RR461. 1986.

G. Geist, C. Romine: "LU Factorization Algorithms on Distributed Memory Multiprocessor Architectures". Oak Ridge National Laboratory Report ORNL/TM-10383. 1987.

M. Heath, C. Romine: "Parallel Solution of Triangular Systems on Distributed Memory Multiprocessors". SIAM J. Sci. Stat. Comp. 9 (1988).

C. Ashcroft, S. Eisenstat, J. Liu , H. Sherman: "A Comparison of Three Column-Base Distributed Sparse Factorization Schemes". Yale University Research Report. YALEU/DCS/RR 810, 1990.

A. George, J. Liu, E. Ng: "Communication Results for Parallel Sparse Cholesky Factorization on a Hypercube". Parallel Computing 10 (1989).

M. Heath, E. Ng, B. Peyton: "Parallel Algorithms for Sparse Linear Equations". SIAM Review 33(1991).

Tools for Parallelism

The objective of the course is to study the kind of tools a parallel software developer has available to him and how much of the parallelizing effort he can hope to have the tools do

for him.

The course has a seminar format. The instructor updates a rather large number of published papers which is distributed amongst the participants for class presentation and discussion. Such seminar classes should be complemented wherever possible by actual hands on experience with representative tools.

Topics

- Vectorizing and parallelizing compilers.
- The mapping problem and granularity.
- Basic communication libraries for distributed memory systems.
- Debugging parallel programs for correctness and for performance.
- Putting it all together: Parallel programming environments

References

Due to the course's seminar structure, the list of readings is constantly changing. The current set of kernel references is:

H. Zima, B. Chapman: "Supercompilers for Parallel and Vector Machines". Addison-Wesley, 1991.

M. Wolfe: Optimizing Supercompilers for Supercomputers". The MIT Press, 1990.

L. Jamieson, D. Gannon, R. Douglass: "The Characteristics of Parallel Algorithms". MIT Press, 1987.

D. Reed, R. Fujimoto: "Multicomputer Networks: Message-Based Parallel Processing". MIT Press, 1987.

C. McDowell, D. Helmbold: "Debugging Concurrent Programs". ACM Computing Surveys. Vol.21 N°4. December 1989.

M. Reilly: "A Performance Monitor for Parallel Programs". Academic Press, 1990.

G. Fox, M. Johnson, G. Lyzenga, J. Salmon, D. Walker: "Solving Problems on Concurrent Processors: Vol. 1". Prentice-Hall, 1988. Chap. 14: CrOS; Chap. 15: CUBIX;

An anthology of twenty to thirty articles is being compiled for the 1992 course. Due to space restrictions they will not be cited in this paper. The interested reader may write directly to the authors for details.

Theory of Parallel Processes

Efficiently executing a parallel program that produces the wrong results or deadlocks is utterly useless. It is important to understand the underlying theory of parallel processes in order to be able to reason about parallel programs in order to prove their correctness or to show whether certain properties like absence of deadlock, livelock or critical races hold. The key to the conceptual basis for reasoning about parallelism lies in the relationship between concurrency and non-determinism. The course covers two of the many theories explicitly developed to understanding such questions and shows how these theories can

be used to design and reason about parallel algorithms and languages.

Topics

- Introduction.

  The need for a formal theory of parallel processes. Mathematically, what is a process? Varieties of models and theories. Concurrency and non-determinism as key issues.

- The theory of UNITY.

  A logic for parallelism. Architectures and mappings. Program compositions. Examples and applications.

- The theory of CSP.

  Events and processes. CSP operators. CSP's models: traces, failures, divergences. Superimposed obligations. Examples and applications. CSP and Occam.

References

C. Hoare: "Communicating Sequential Processes". Prentice-Hall, 1985.

K. Misra, J. Chandy: "Parallel Program Design: A Foundation". Addison-Wesley, 1988.

Experience and conclusions

Our experience in setting up and tuning a course of studies in Parallelism has been extremely interesting and very rewarding, in that it has forced us to attempt to integrate and clarify a great deal of scattered material and subtle interplays of software and hardware. We would like to enrich the actual parallel environment so as to provide more hands-on experience and strengthen the coverage of performance modelling techniques, especially as regards statistical modelling techniques. While it is clearly necessary to provide in-depth coverage of representative theories on parallelism in order to appreciate their strengths, weaknesses and subtleties, restricting such a study to two such theories may seem unduly constrictive --in particular, omitting any reference to Petri Net based theories is definitely worrisome. In fact, striking the right balance between depth of treatment and breadth of coverage is a recurring thread throughout the evolution of all our courses.

Student and industry feedback has been very positive, especially as regards the Venezuelan oil industry whose keen interest in the venture has been a constant spur to our efforts. In particular, we would like to thank Alfonso Reinoza, Jorge Baralt, Oscar Meza and Cristina Zoltán of the USB, Raju Karia, Benjamín Sagalovsky, Saúl Buitrago and Enrique Rodríguez of INTEVEP, Rubén Amaya of MARAVEN, Walter Cunto, Kley Visentín and Jorge Gonçalves of IBM Venezuela, Jorge Vidart of ESLAI, Euripides Montagne of the Universidad Central de Venezuela and, of course, all our students, whose comments and enthusiasm have helped shape our programme.

DISPARITY-BASED PHOTOMETRIC STEREO

José R.A. Torreão and Cecilio J.L. Pimentel
Departamento de Informática, Universidade Federal de Pernambuco
50.739 Recife PE Brasil

Abstract

We point out a relationship between the processes of Stereoscopy and Photometric Stereo which can be exploited for the estimation of depth with single-camera imaging set-ups.

The displacement of pixel intensities observed when the illumination direction changes, in the formation of photometric stereo image pairs, can be treated as a disparity map arising in an equivalent convergent stereoscopic system. Such photometric disparity map can thus be estimated through the use of a stochastic approach which essentially matches the two images at pixel level, while the shape of the pictured surface can be simultaneously reconstructed by the usual photometric stereo technique. From the disparity and shape information, the parameters of the equivalent stereoscopic system can be estimated, and a depth field can thus be recovered by triangulation. This strategy allows the simultaneous shape and depth reconstruction from photometric stereo image pairs, which could be useful for simple industrial vision applications. Preliminary experimental results obtained with such scheme, for synthetic images, are presented.

KEYWORDS: Artificial Vision, Stereoscopy, Photometric Stereo

The present work has been supported by the grant APQ 171-1.03/90 from FACEPE (Fundação de Amparo à Ciência e Tecnologia do Estado de Pernambuco).

1. Introduction

Stereoscopy and Photometric Stereo are two processes which are widely used for the estimation of geometric properties of imaged surfaces in Computational Vision. The first, also known as Geometric Stereo [**Grimson 86, Barnard 82**], extracts depth information from the displacement of features (which is called 'disparity') observed when two images of a scene are captured from different viewing positions. At the core of that process is the matching between points in the two images which are projections of the same location in the scene. Photometric Stereo [**Woodham 80**], on the other hand, is a Reflectance-Map technique employing two or more images, obtained from a single camera for different illumination directions, which allows the estimation of an orientation map for the viewed surfaces. The relevant information, in this case, is yielded by the multiple intensity values at each pixel site.

There is an obvious relationship between the two stereo modalities, since they are both multiple-image processes which yield the related properties of depth ($z(x,y)$) and surface orientation ($\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}$), but, beyond this, there is a less apparent and more useful duality between the two, as yet not fully exploited [**Torreão 91**]. This arises from the fact that in Stereoscopy, as well as in Photometric Stereo, both the geometric information (displacement of features) and the photometric information (multiple intensities) are available to be used for surface estimation purposes.

In Stereoscopy, for instance, once the correspondence of a pair of points in the images has been established, there will be two intensity values available, in the general case, for the recovery of surface orientation, once the Reflectance Map [**Horn 77, 79**] is known. On the other hand, for smooth surfaces, the change of illumination direction in Photometric Stereo entails a variation of pixel intensities in the image lattice, which can be treated as an intensity-displacement field akin to the disparity field resulting from the change of viewing position in Stereoscopy. Such displacement field (which will be referred to as the *photometric disparity* field) can, for example, be obtained through the stochastic algorithm for stereo correspondence proposed by Barnard [**Barnard 86**], where the matching between images is performed essentially at pixel level, yielding a dense map.

If we can determine a stereoscopic configuration which yields a disparity map similar to the one obtained from a given pair of photometric stereo images, we can employ such map for the inference of depth via simple triangulation. Here we propose a strategy for finding such equivalent (convergent) stereo system, when the displacement field and the surface shape have been recovered from a given photometric stereo pair. Our strategy is applicable for smooth surfaces, when the angle between

the illumination directions under which the images are captured is small. Under such conditions it allows the use of a single-camera set-up for the essentially simultaneous estimation of both depth and orientation. This could prove to be a very attractive alternative to the use of more expensive stereoscopic arrangements (involving Dual Photometric Stereo [**Ikeuchi 87**], for instance) in simple industrial vision applications.

2. Relating Photometric Stereo and Stereoscopic Systems

We refer to Figure 1, where a cross-sectional view of an imaging arrangement consisting of two cameras in a convergent stereoscopic configuration is shown. The coordinate axes associated with the scene and with each camera are denoted respectively by (x, z), (x_1, z_1), and (x_2, z_2), with the corresponding y-axes (not shown) being perpendicular to the paper. It should be noted that, for convenience, the axis z, in the scene system, points in the opposite direction to the axis z_1 in the first-camera system. With such orientation choice, the coordinates of a point, as given in these two systems, are the same. Two illumination directions and two viewing directions are represented, denoted respectively by $\hat{s}$ and $\hat{s}'$, and by $\hat{v}$ and $\hat{v}'$. Observe that the vectors $\hat{v}'$ and $\hat{s}'$ are obtained as rotations, in the xz plane, of the vectors $\hat{v}$ and $\hat{s}$, respectively. It is evident that, for such stereo configuration, the epipolar lines will lie along the x-direction. (We recall that, given a scene point, the associated epipolar lines in the images are those lines where the plane containing such point and the centers of projection of the two cameras cut the image planes. The search for matching pairs of points in the stereoscopic images needs to be performed only across the epipolar lines.)

We will be considering two pairs of images obtained with such imaging arrangement. These correspond to a stereoscopic pair obtained from the viewing directions $\hat{v}$ and $\hat{v}'$, under illumination $\hat{s}$, and a photometric stereo pair obtained from the viewing direction $\hat{v}$ under the illuminations $\hat{s}$ and $\hat{s}'$. The two pairs, therefore, share a common image: the one obtained from $\hat{v}$ under illumination $\hat{s}$. With the use of the reflectance map function R [**Horn 77, 79**], those images can be expressed as

$$I_1(i,j) = R(\hat{n}_{ij}, \hat{s}, \hat{v}),\ I_2^S(i, j - D_{ij}) = R(\hat{n}_{ij}, \hat{s}, \hat{v}'),\ \text{and}\ I_2^P(i,j) = R(\hat{n}_{ij}, \hat{s}', \hat{v}), \qquad (1)$$

where D_{ij} and $\hat{n}_{ij}$ denote the disparity and the surface-orietation normal vector at site (i, j), which is given in the scene coordinate system, and where the superscripts S and P stand for stereoscopic and photometric stereo, respectively.

Applying the stochastic stereo algorithm by Barnard [**Barnard 86**], we can extract disparity

maps from each of the image pairs by minimizing, via simulated annealing relaxation [Kirkpatrick 83, Geman 84], a cost function of the form

$$E = U + \lambda V, \tag{2}$$

with

$$U = \sum_{i,j} |I_1(i,j) - I_2(i, j + D_{ij})|, \tag{3}$$

and

$$V = \sum_{i,j} |\nabla D_{ij}|. \tag{4}$$

The term U represents a *photometric constraint*, which dictates that the disparity assignments should map points in the first image to points in the second image with comparable intensity, while V encodes a *smoothness constraint*, which penalizes disparity maps with abrupt local variations, and which is implemented as the sum of the absolute differences between the disparity at site (i,j) and those on its eight neighboring sites. The constant λ is a weighting factor which determines the relative contribution of the two terms to E.

We now recall that our aim is to find a stereoscopic configuration which yields a pair of images similar to a given photometric stereo pair, in the sense that the intensity displacement arising from the change of viewing position, in the first case, should be the same as the one arising from the illumination change, in the second. By considering the form of equation (3) for the image pairs $\{I_1, I_2^S\}$ and $\{I_1, I_2^P\}$, when I_1, I_2^S and I_2^P are given in terms of the reflectance maps as in equation (1), it is easy to see that similar photometric and stereoscopic disparity maps will be obtained when the direction $\hat{v}'$ satisfies the relation

$$R(\hat{n}_{ij}, \hat{s}, \hat{v}') \approx R(\hat{n}_{i,j+D_{ij}}, \hat{s}', \hat{v}), \quad \forall i,j. \tag{5}$$

We thus suggest the following strategy for the parallel estimation of depth and shape in a photometric stereo configuration:

1) Barnard's algorithm can be employed for extracting a photometric disparity map through the minimization of the cost functional

$$E(D) = \sum_{ij} |I_1(i,j) - I_2^P(i, j + D_{ij})| + \lambda \sum_{ij} |\nabla D_{ij}|, \tag{6}$$

via stochastic relaxation, while shape estimates can be obtained through the traditional photometric stereo reconstruction.

2) From the results for $\hat{n}_{ij}$ and D_{ij}, the equivalent stereoscopic direction $\hat{v}'$ can be determined from equation (5) by the minimization of

$$E(\hat{v}') \equiv E(\theta_s) = \sum_{ij} |R(\hat{n}_{ij}, \hat{s}, \hat{v}') - R(\hat{n}_{i,j+D_{ij}}, \hat{s}', \hat{v})|, \tag{7}$$

where, from Figure 1, we see that $\hat{v}'$ has the form $\hat{v}' = (\sin\theta_s, 0, \cos\theta_s)$.

3) Once the angle θ_{eq} which minimizes the expression in (7) ($\theta_{eq} = \arg\min\{E(\theta_s)\}$) is available, relative depth estimates can be obtained from the photometric disparity field through the relation

$$z(i,j) = \frac{1}{\sin\theta_{eq}}[D_{ij} - X(i,j)(1 - \cos\theta_{eq})],$$

where we have taken $d = z_{max}\tan\theta_{eq}$ for the camera separation, and where $z(i,j) = z_{max} - z_{abs}(i,j)$ (see Figure 1).

In the following section, we present results of the aplication of the above strategy to some pairs of synthetically generated images.

3. Obtaining Shape and Depth from the Disparity-Based Photometric Stereo

We have considered Reflectance Maps as below [**Horn 77**], involving both a diffuse reflection component, and a quasi-specular component given by the Phong model [**Phong 75**],

$$R(\hat{n}, \hat{s}, \hat{v}) = \rho[(1-\sigma)\hat{n}.\hat{s} + \frac{1}{2}\sigma(m+1)(2(\hat{n}.\hat{s})(\hat{n}.\hat{v}) - \hat{v}.\hat{s})^m], \tag{8}$$

where ρ is the albedo, σ is the specular weight, and m is a constant. When $\sigma = 0$, and therefore only the diffuse or Lambertian reflection is present, the Reflectance Map carries no information about the observation direction, and therefore equation (7) can not be used for the estimation of $\hat{v}'$. We remark that we are, nevertheless, still able to employ our strategy in such case, by considering equation (5) for a Reflectance Map as in (8) in the limit of $\sigma \to 0$. In such limit, the minimization of (6) matches points for which $\hat{n}_{ij}.\hat{s} \approx \hat{n}_{i,j+D_{ij}}.\hat{s}'$. Employing this relation, together with the form (8), in equation (5), it is easy to obtain the condition $(2(\hat{n}_{ij}.\hat{s})(\hat{n}_{ij}.\hat{v}') - \hat{v}'.\hat{s}) \approx (2(\hat{n}_{i,j+D_{ij}}.\hat{s}')(\hat{n}_{i,j+D_{ij}}.\hat{v}) - \hat{v}.\hat{s}')$, for all sites (i,j). We can thus use for $E(\hat{v}')$, in the Lambertian case, the expression

$$E(\hat{v}') \equiv E(\theta_s) = \sum_{ij} |(2(\hat{n}_{ij}.\hat{s})(\hat{n}_{ij}.\hat{v}') - \hat{v}'.\hat{s}) - (2(\hat{n}_{i,j+D_{ij}}.\hat{s}')(\hat{n}_{i,j+D_{ij}}.\hat{v}) - \hat{v}.\hat{s}')|, \tag{9}$$

which is a purely geometric relation, independent of σ.

The photometric stereo images employed in our experiments have been generated through the use of the Image Irradiance Equation [Horn 77, 79]

$$I^{(k)}(i,j) = R(\hat{n}, \hat{s}_k, \hat{v}), \quad k = 0, 1, 2$$

for Lambertian reflectance maps, and illumination directions $\hat{s}_0 = (0, 0, 1)$, $\hat{s}_1 = (\sin\theta_p, 0, \cos\theta_p)$ and $\hat{s}_2 = (\sin(-\theta_p), 0, \cos(-\theta_p)) \equiv (-\sin\theta_p, 0, \cos\theta_p)$. From the pairs of images $\{I^{(0)}, I^{(1)}\}$ and $\{I^{(0)}, I^{(2)}\}$, two disparity fields, $D_1(i,j)$ and $D_2(i,j)$ (which are taken here in absolute value), were obtained. From these, a combined photometric disparity map was constructed as $D(i,j) = D_1(i,j) - D_2(i,j)$, which, together with the surface orientation estimates yielded by photometric stereo, has been employed for the determination of the equivalent stereoscopic angle θ_{eq}, through (9). We have used this combination of disparity maps obtained for illumination directions symmetric about the z-axis, in order to avoid the unmatchable regions which occur when only one pair of photometric stereo images is employed (such regions, where no disparity information is available, are the equivalent of the occluded areas for stereoscopic systems). Depth estimates can be obtained from such combined maps, as below:

As we have seen, relative depth can be expressed in terms of disparity, for the equivalent convergent stereoscopic system, by (see Figure 1)

$$z(i,j) = \frac{D_1(i,j) - (1 - \cos\theta_{eq}).X(i,j)}{\sin\theta_{eq}}.$$

For an angle $-\theta_{eq}$, this expression becomes

$$z(i,j) = \frac{-D_2(i,j) + (1 - \cos\theta_{eq}).X(i,j)}{\sin\theta_{eq}},$$

and from the two expressions above we can get

$$z(i,j) = \frac{D_1(i,j) - D_2(i,j)}{2.\sin\theta_{eq}} = \frac{D(i,j)}{2.\sin\theta_{eq}}, \tag{10}$$

which is the relation we have employed in our estimation of depth from the photometric disparity data.

In figures 2 through 4, we show results of depth reconstruction from the above scheme. Letters (a) in those figures depict input photometric stereo pairs (for angles θ_p equal to 15°, 10°, and 12°, respectively), along with the recovered disparity fields. Letters (b) show the respective tridimensional depth reconstructions obtained from (10), where the angles θ_{eq} have been estimated from

the curves in Figure 5, which represent the form of $E(\theta_s) \times \theta_s$ for each of the imaged surfaces. From Figure 5, we find the minimizing angles θ_{eq} of 15°, 10°, and 20°, respectively, and we observe that, for surfaces which show axial symmetry about the y-axis, one has simply $\theta_{eq} = \theta_p$, as should have been expected [Torreão 91]. The depth values obtained in those reconstructions have been consistent with the synthetic surfaces employed.

4. Concluding Remarks

In this article, we have proposed a single-camera depth evaluation scheme, based on an analogy between the processes of Stereoscopy and Photometric Stereo, which allows the extraction of a disparity map from a photometric stereo image pair. Such disparity map, together with an orientation field obtained from the traditional photometric stereo technique, can be employed for the determination of the vergence angle of an equivalent convergent stereoscopic set-up. In this way, depth estimates can be recovered from the obtained photometric disparity field. Here we have considered only the estimation of relative depth, but we are presently trying to extend our framework for the treatment of images generated under a perspective projection geometry, which would allow the inference of absolute depth information.

5. References

[Barnard 86] Barnard, S.T., *A Stochastic Approach to Stereo Vision*, in Procs. 5th National Conf. on AI, pp. 676-680, Philadelphia, PA, 1986.

[Barnard 82] Barnard, S.T., and M.A. Fischler, *Computational Stereo*, Computing Surveys, vol.14(4), pp. 553–572, 1982.

[Geman 84] Geman S., and D. Geman, *Stochastic Relaxation, Gibbs Distributions, and The Bayesian Restoration of Images*, IEEE Trans. Patt. Anal. Mach. Intell. vol.6, pp. 721–741, 1984.

[Grimson 86] Grimson, W.E.L., *From Images to Surfaces*, The MIT Press, Cambridge, MA, 1986.

[Horn 77] Horn, B.K.P., *Understanding Image Intensities*, Artificial Intell., vol.8(11), pp. 201–231, 1977.

[Horn 79] Horn, B.K.P., and R.W. Sjoberg, *Calculating the Reflectance Map*, Applied Optics,

vol.18(11), pp. 1770–1779, 1979.

[Ikeuchi 87] Ikeuchi, K., *Determining a Depth Map Using a Dual Photometric Stereo*, Int. J. Robotics Research, vol. 6(1), 1987.

[Kirkpatrick 83] Kirkpatrick, S., C.D. Gelatt, Jr., and M.P. Vecchi, *Optimization by Simulated Annealing*, Science, 220(4598), pp. 671–680, 1983.

[Phong 75] Phong, B.T., *Illumination for Computer Generated Pictures*, Commun. Ass. Comput. Mach., vol. 18(6), 1975.

[Torreão 89] Torreão, J.R.A., *A Bayesian Approach to 3-D Shape Estimation for Robotic Vision*, Ph.D. Thesis, Department of Physics, Brown University, September 1989.

[Torreão 90a] Torreão, J.R.A., *Novel Applications of the Photometric Stereo Set-Up for Robotic Vision*, in Procs. X Congress of the Brazilian Computing Society, Vitória, Brazil, July 1990.

[Torreão 90b] Torreão, J.R.A., *A Bayesian Framework for the Estimation of Shape and Shape Discontinuities in 3-D*, in Procs. XVI Latin-American Conference in Informatics – CLEI, Asunción, Paraguay, September 1990.

[Torreão 91] Torreão, J.R.A., *Towards a Single-Camera Stochastic Stereoscopy*, in Procs. VIII Brazilian Symposium on Artificial Intelligence, Brasília, Brazil, November 1991.

[Woodham 80] R.P. Woodham, *Photometric Method for Determining Surface Orientation from Multiple Images*, Optical Engineering, vol.19(1), pp. 139–144, 1980.

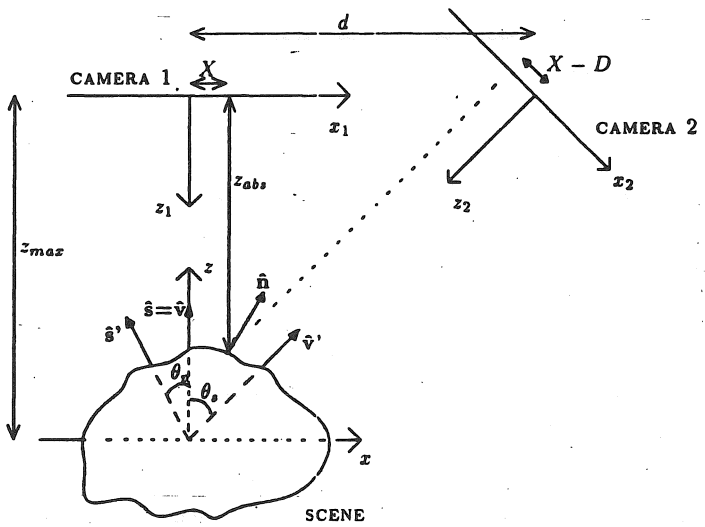

d
CAMERA 1
X
X − D
CAMERA 2
x_1
x_2
z_1
z_2
z_{abs}
z_{max}
z
$\hat{n}$
$\hat{s}=\hat{v}$
$\hat{s}'$
$\hat{v}'$
θ_i
θ_e
x
SCENE

Figure 1

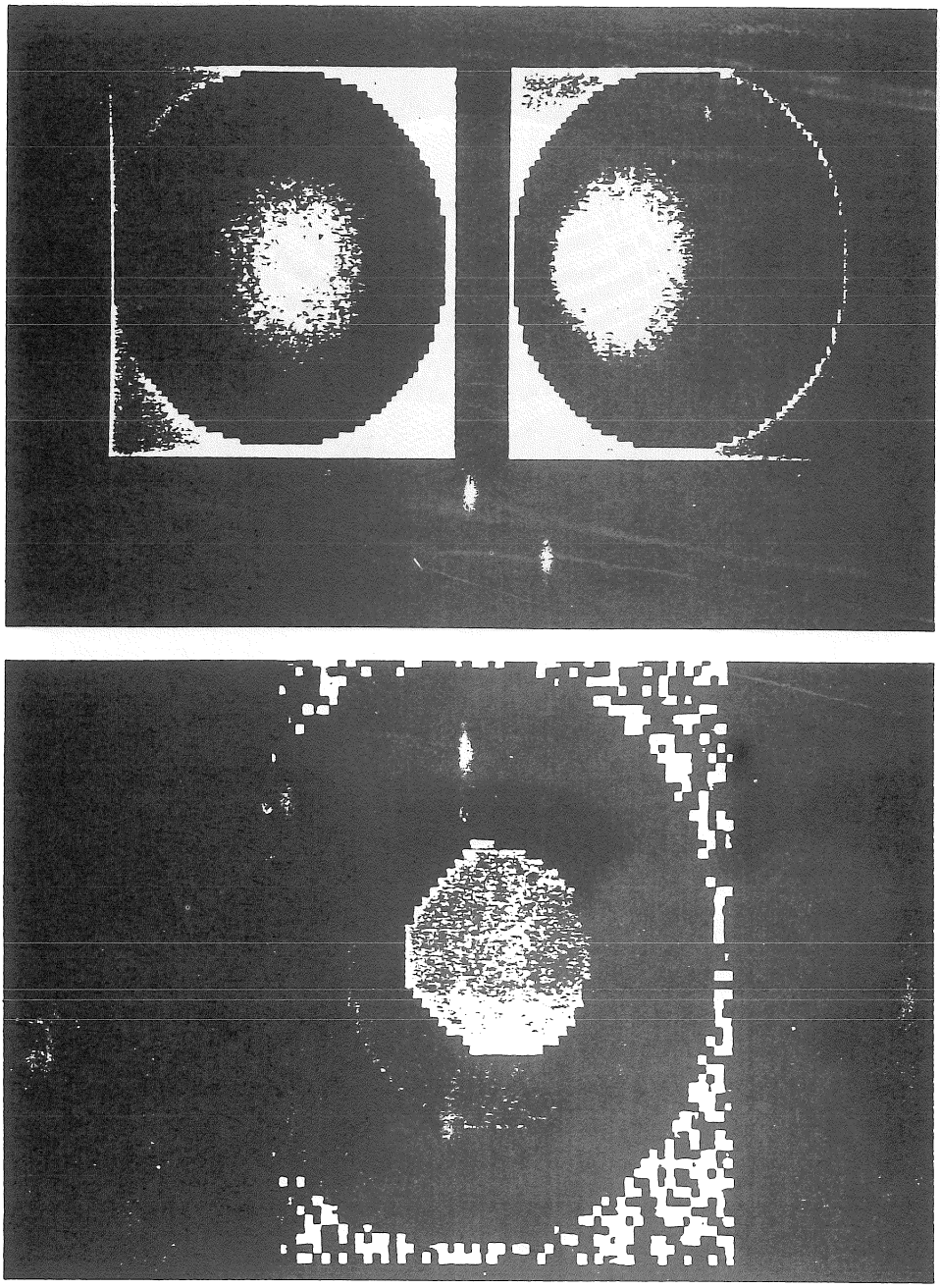


Figure 2a

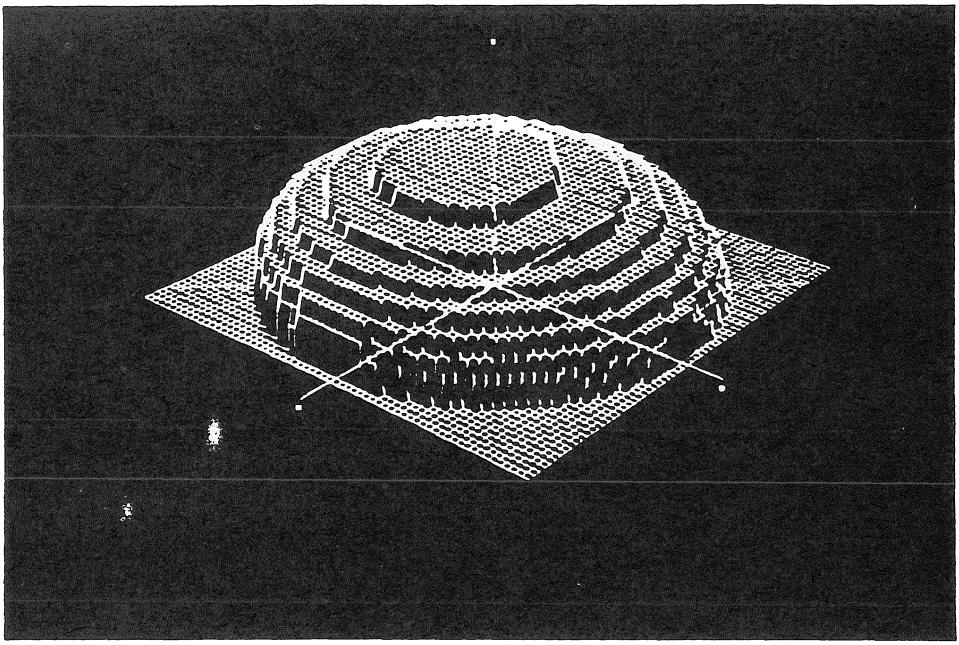


Figure 2b

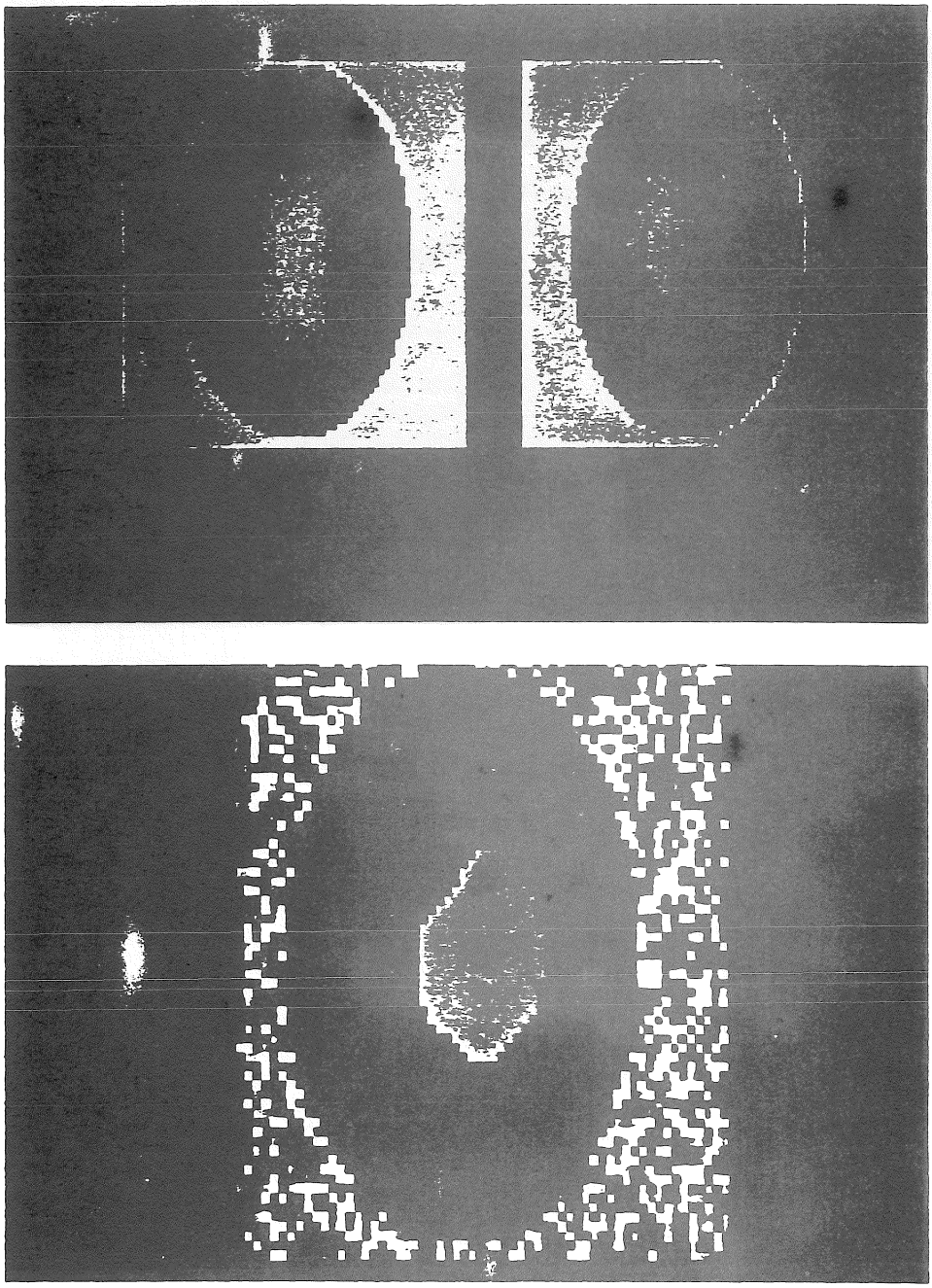


Figure 3a

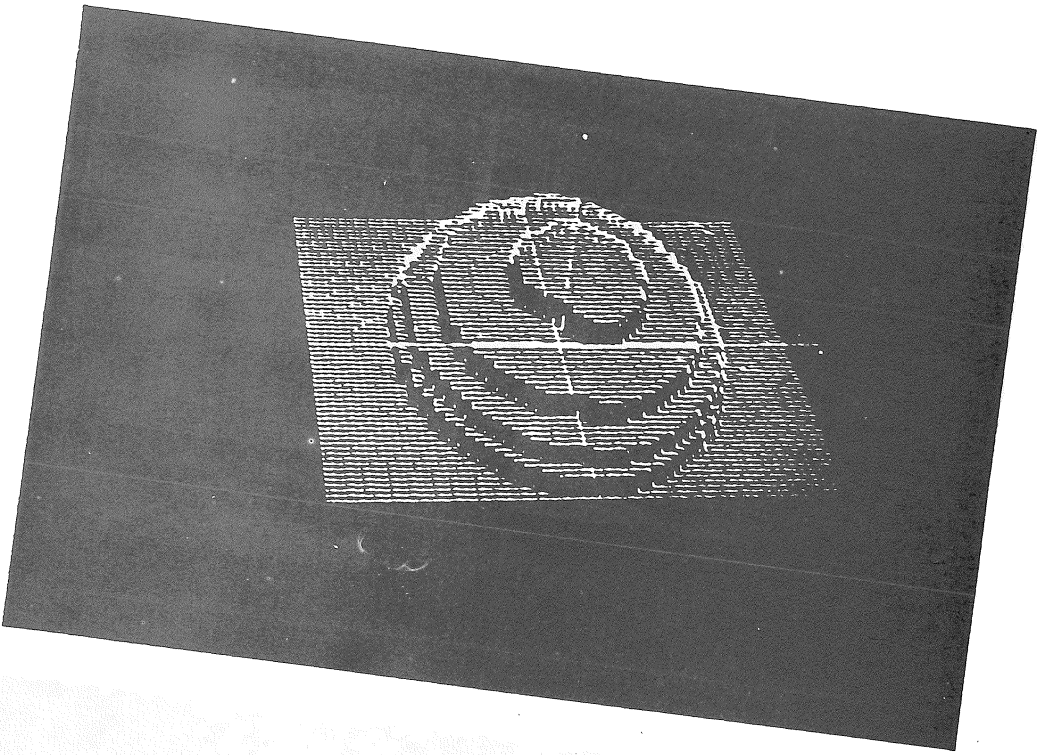


Figure 3b

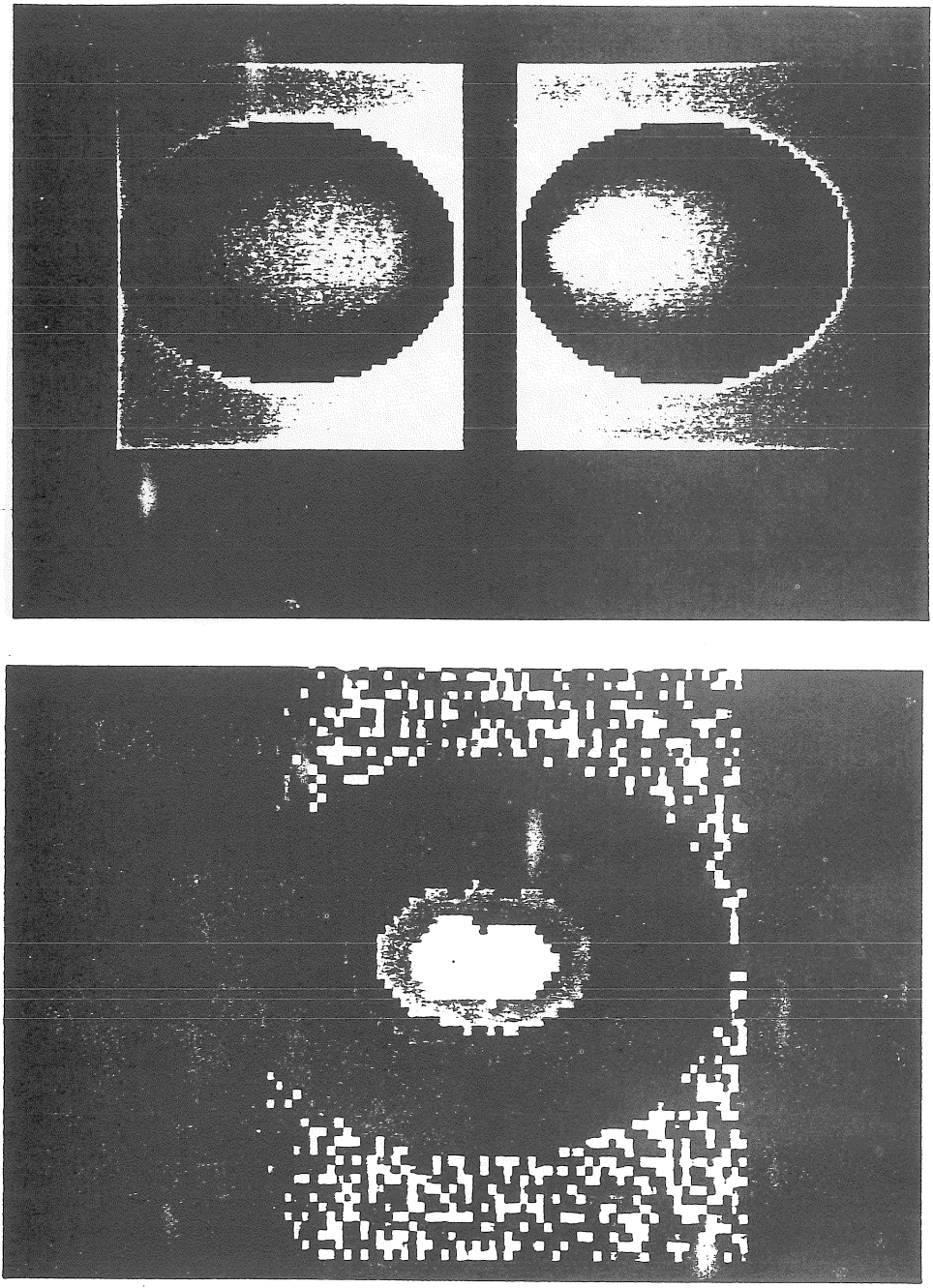


Figure 4a

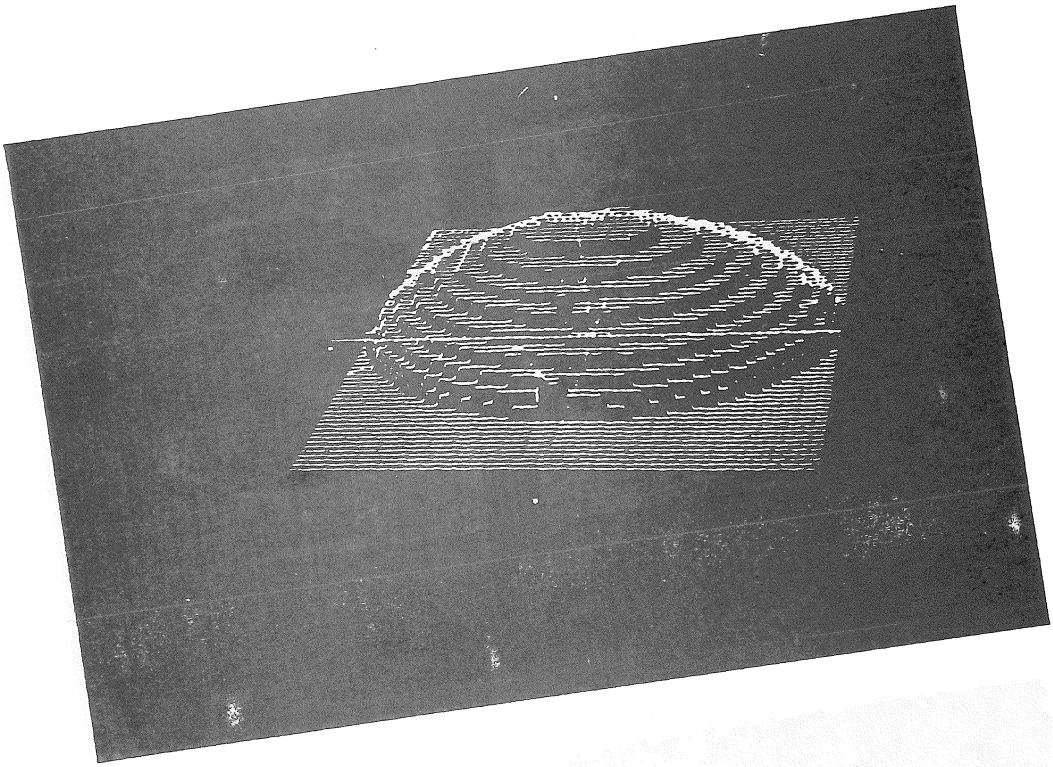


Figure 4b

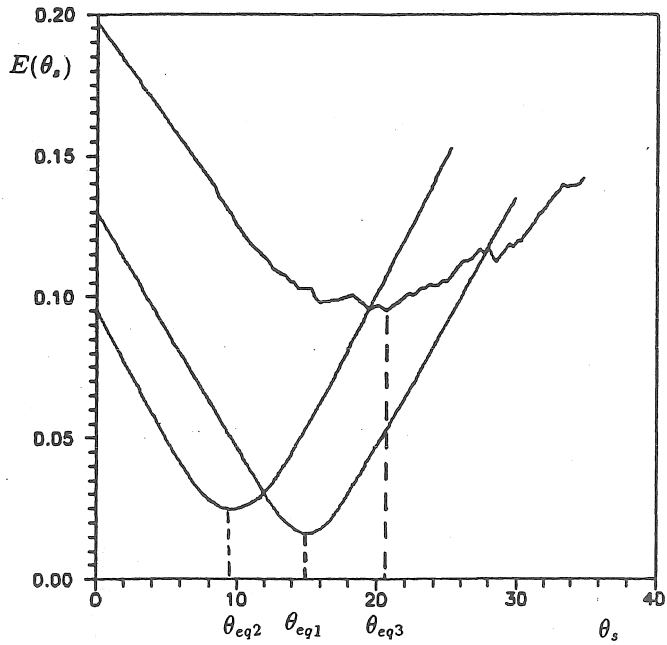

0.20
$E(\theta_s)$
0.15
0.10
0.05
0.00
0
10
20
30
40
θ_{eq2}
θ_{eq1}
θ_{eq3}
θ_s

Figure 5

TECNOLOGIA DE INTELIGENCIA ARTIFICIAL EN LAS ARQUITECTURAS PARA SISTEMAS DE CONTROL

Irene Torres Hecker. Paradigma C.A.

Apdo. Postal 1087. Ciudad Guayana 8015. Venezuela

ABSTRACT

La aplicación de nuevas tecnologías, en particular, de Inteligencia Artificial, para abordar problemas de Control en ambiente industrial ha demostrado ser altamente provechosa, encontrándose en la práctica esquemas consolidados como el uso de Sistemas Expertos, y otros aún con caracter experimental, como la integración de módulos de procesamiento mediante Redes Neurales en distintos puntos de las arquitecturas de control.

Existen aspectos propios del proceso, así como de los recursos disponibles, que son determinantes para un diseño adecuado. La identificación de dichos factores a la luz de las nuevas tecnologías a aplicarse, en particular aquellas del campo de la Inteligencia Artificial darán garantía de resultados exitosos.

KEYWORDS: Control de Procesos, Arquitecturas de Control, Sistemas Expertos

APLICACION DE INTELIGENCIA ARTIFICIAL EN ARQUITECTURAS DE CONTROL

Dados los niveles de **Control Local** y **Control Supervisorio**, en los cuales se verifican ciclos de **adquisición de datos**, **almacenamiento**, **análisis** y **corrección**, y que en conjunto con el **Control Estratégico** caracterizan los ambientes industriales, existen distintas arquitecturas considerando por un lado el grado de acoplamiento entre los componentes -impuesto por los requerimientos Tiempo Real de los procesos a controlar- y por otro lado, propiamente los componentes que integran la arquitectura.

En todos los planteamientos de arquitecturas, de **Acoplamiento Débil** o **Fuerte**, y en relación a su naturaleza de **Control Local** o **Supervisorio**, con el consecuente grado de intervención humana en su operación, se distinguen dos componentes siempre presentes: un Módulo de Adquisición de Datos y un Módulo de Análisis. El primero de ellos se beneficia de los adelantos principalmente en el campo de la electrónica y se complementa con el desarrollo de software adecuado para hacer uso de la circuitería en cuestión. Con esto se cubren las variantes de captores lógicos y captores físicos en el nivel más básico y por otra parte la construcción de interfaces cada vez más poderosas para permitir que los mecanismos de adquisición de datos incorporen locaciones remotas, esquemas distribuidos o distintos componentes integrados en redes.

En cuanto al proceso de análisis sobre los datos adquiridos, éste puede variar desde formas muy elementales en las que simplemente se apliquen criterios de organización a los datos hasta formas que incorporen un alto grado de "inteligencia". La opción escogida se deberá en gran parte a factores tales como el propósito de la respuesta a alcanzar, la dinámica del proceso a controlar, o incluso el riesgo implícito en el mismo.

Un gran número de procesos industriales tienen un comportamiento Tiempo Real bastante estricto, lo que, en conjunto con una estabilidad vulnerable del proceso, obliga a que los sistemas de control busquen incorporar, cada vez con mayor énfasis, las siguientes características:

* velocidad de procesamiento -análisis- a la par de la velocidad natural del proceso
* capacidad para almacenar y operar grandes volúmenes de datos
* integración de distintas formas de procesamiento: procedimental, inferencial
* respuestas de distinta naturaleza: refleja, interpretativa, predictiva
* capacidad de enfrentar casos excepcionales

Dentro de diversas áreas de la Inteligencia Artificial se encuentran paradigmas capaces de ofrecer respuesta a las demandas anteriormente mencionadas. A continuación se examinarán algunas de ellas y se expondrán ejemplos de su aplicación.

Sistemas Basados en Conocimiento:

Bajo el término Sistemas Basados en Conocimiento se considerarán los Sistemas Expertos y los Sistemas de Simulación Basada en Conocimiento. En los Sistemas Expertos que se desarrollan en la actualidad existe un uso generalizado del Enfoque de Objetos como forma de representación del conocimiento e igualmente una incorporación de facto de capacidades para el procesamiento requerido por Sistemas de Control, que se traduce en poderosas interfaces hacia equipos de medición y adquisición de datos y en manejo natural de conceptos de Tiempo Real.

En el marco de las arquitecturas de control los sistemas expertos pueden constituir módulos de los sistemas para el análisis de los datos, ofreciendo la posibilidad de evaluar la conducta del proceso controlado en términos cualitativos con la incorporación de un volumen de información mucho mayor al manejable por un experto humano y en un tiempo más corto, predecir eventos basados en las tendencias observadas en los parámetros de control, diagnosticar eventos excepcionales, coordinar múltiples modos de respuesta seleccionando entre respuestas reactivas en condiciones de normalidad y respuestas "razonadas"en situaciones problema y redefinir el curso de acción del proceso, entre otras bondades.

En cuanto a las variantes para la inserción de módulos expertos, determinadas por la arquitectura elegida, cabe mencionar que en los casos en que el análisis se hace fuera de línea, y en los casos en que se cuenta con la intervención de un operador, la naturaleza del módulo experto puede ser de Recomendación o Asistencia. En otro caso, en el que es responsabilidad del propio sistema la aplicación de sus resultados al proceso, su naturaleza es de Acción.

La utilización de Sistemas Expertos para una preclasificación de las entradas recibidas para el análisis y la orientación a priori del curso del mismo, es otra propuesta de inclusión de Módulos Expertos en Arquitecturas de Control. Entre las limitaciones que se enfrentan con esta concepción está la de tener que ser exhaustivo en la consideración de los casos a ocurrir.

Los Sistemas de Simulación Basada en Conocimiento ofrecen la posibilidad de modelar de forma cualitativa, en combinación con otras formas convencionales, los procesos a controlar y obtener una representación de los mismos muy cercana a la realidad. Esto permite pronosticar la ocurrencia de eventos y tomar previsiones fundamentadas. Por otra parte la utilización de módulos de Simulación de esta naturaleza constituye un gran aporte a la labor de interpretación de los resultados que se obtienen de la misma porque ellos son utilizados por retroalimentación como insumo para ajustes al modelo originalmente concebido.

La inserción de módulos de Simulación Basada en Conocimiento tiene cabida como recurso para complementar el procesamiento a nivel de Control Supervisorio. Un Elemento Simulador puede, entre otras cosas, suplir datos ante fallas en la adquisición real de los mismos y de esta manera permitir que el sistema de control funcione ininterrumpidamente. También permite estudiar el efecto de posibles acciones en un plazo más largo.

Redes Neurales:

Las Redes Neurales, con sus esquemas de procesamiento distribuido y paralelo, han encontrado un amplio terreno de aplicación dentro del contexto industrial por su uso para el procesamiento de patrones -visto el reconocimiento como parte de esta tarea-.

En el caso de los Sistemas de Control de Lazo Cerrado, en los que tiene lugar el ciclo de Adquisición de Datos, Análisis, y Modificación, se presenta típicamente un caso de reconocimiento de patrones, en el cual estos últimos están formados por vectores de los valores adoptados por los distintos parámetros del proceso elegidos para el control, y en el cual el

sistema efectúa una clasificación de los patrones hacia una situación del proceso y en consecuencia produce una salida que puede resultar un mapeo de la clasificación obtenida, hacia las acciones que conviene ejecutar.

La clasificación de patrones se puede ver como una regla de decisión que se especifica a través de la selección de valores para un conjunto de parámetros. El aprendizaje es entonces el proceso de ajuste de los valores de los parámetros basado en qué tan bien se comporta la regla de decisión sobre un conjunto de patrones de entrenamiento cuyas clasificaciones correctas se suplen.

El proceso anterior, en un enfoque conexionista puede ser la creación de memorias asociativas, en donde los pesos de las conexiones se ajustan por Estimación de Parámetros, en base a un error que se minimiza incrementalmente y el proceso es más que un ajuste del modelo, un registro de información en una "memoria".

La aplicación de Redes Neurales de la forma descrita, con su procesamiento distribuido y paralelo, a problemas de Control encuentra justificación por los siguientes factores:

* Los procesos que se controlan son dinámicos, dependientes de su historia, por lo cual un modelo con mapas estáticos de I/O resulta inadecuado
* El desempeño del sistema debe ajustarse en varios ciclos de control, por un proceso de Entonación, y no se determina por respuestas instantáneas
* Existen requerimientos de adaptación y aprendizaje para el Control en Lazos Cerrados
* La información proveniente de procesos puede ser ruidosa o incompleta y sin embargo se requiere respuesta por parte del sistema

Por otra parte, existe un conjunto de aspectos esenciales para la adecuación de una Red Neural al control de un proceso. Ellos son:

* representación del problema: se refiere a la selección de las variables del proceso, a los datos que se adoptarán para la clasificación y construcción del modelo, y al proceso de codificación que se operará sobre los valores
* diseño de la red: considera la utilización de los parámetros seleccionados, las unidades en que serán manejados, las relaciones entre ellos, la estructuración de la red, las restricciones sobre las conexiones y demás aspectos referidos a la instrumentación de las reglas de decisión que definen el modelo
* criterios de desempeño: está referido a la definición de medidas elocuentes, de fácil adquisición que permitan establecer una métrica para la evaluación del desempeño de la red sin que haya que implementar completamentamente cada modelo
* entrenamiento: combinar la posibilidad de entrenamiento fuera de línea con el entrenamiento en línea paralelo al control del proceso
* uso del conocimiento a priori: utilización durante el diseño de la red de la información con que se cuenta en relación a los posibles valores iniciales de los parámetros y las restricciones sobre ellos, para acelerar y mejorar el aprendizaje

Verificando los aspectos anteriormente expuestos, las Redes Neurales pueden ser insertadas dentro de las arquitecturas de Control, para el procesamiento de los datos adquiridos, bien sea para la generación de una respuesta inmediata al proceso u operador, según el esquema en cuestión, o bien para una evaluación preliminar de la situación que luego será aprovechada por otros módulos del sistema de análisis.

Durante el proceso de concepción del Sistema de Control para un proceso, también las Redes Neurales encuentran aplicabilidad, principalmente en lo que se refiere a la **Identificación del Sistema** o en otras palabras la definición del modelo, con la actividad de Detección de Rasgos.

Esquemas Híbridos:

La propuesta básica es la integración de distintas formas de procesamiento para la conformación del módulo de Análisis del Sistema de Control. En particular se consideran diferentes maneras de interacción cooperativa entre Redes Neurales y Sistemas Basados en Conocimiento.

Un ejemplo de este esquema presentado es la utilización de una Red Neural para el Control Local, y la comunicación de sus resultados, complementando los datos directos del proceso, a un módulo Basado en Conocimiento, que integre a su vez capacidades de Simulacion, para el Control Supervisorio. La respuesta del módulo Basado en Conocimiento no estaría limitada a modificaciones al proceso, sino que podría igualmente incluir sugerencias de ajustes al modelo representado por la Red Neural.

Los niveles de **Control Local** y **Control Supervisorio**, con la correspondiente jerarquía de comportamiento definida entre ellos, se integran al nivel de **Control Estratégico**, proporcionándole la información que servirá de insumo para las actividades propias de este nivel, en el cual aparece la Simulación como instrumento de gran valor, principalmente para la evaluación del efecto de cambios previstos. Bajo la óptica ingenieril, para la instrumentación de cambios, la Simulación es un recurso para la ejecución de pruebas que en ambientes industriales, llevadas a escala real, resultarían o sumamente costosas o sumamente riesgosas. Desde la perspectiva gerencial, las bondades de la Simulación han sido tradicionalmente reconocidas.

CONCLUSIONES

* El área de control de procesos -en ambiente industrial en este caso- obtiene un alto beneficio de la inserción de tecnologías de distinta naturaleza, en particular, de la

aplicación de distintos paradigmas de la Inteligencia Artificial para el procesamiento de los datos obtenidos de los procesos controlados.

* El éxito de la aplicación de determinadas tecnologías estará sujeto a las condiciones en que ésta se lleve a cabo, siendo necesarios la consideración de apectos técnicos complejos y un conocimiento bastante sólido tanto del problema a abordar como de los recursos computacionales con los cuales se cuenta.

* Las tendencias en el área de Control de Procesos están dadas por la búsqueda de la integración de los distintos niveles en los que se ejecuta, con la incorporación de tecnologías de punta, para la consecución del objetivo definido como **Control Total** dentro de las organizaciones.

BIBLIOGRAFIA

[1] A. Barr, P. Cohen, E. Feigenbaum. The Handbook of Artificial Intelligence. Vol IV. Addison-Wesley. 1989.

[2] B. Hayes-Roth et al. Intelligent Monitoring and Control. Proceedings IJCAI '89.

[3] B. Kuo. Sistemas Automáticos de Control. Prentice-Hall, Inc. 1962.

[4] B. Silverman. Distributed Inference and Fusion Algorithms for Real-Time Supervisory Controller Positions. IEEE. Knowledge and Data Engineering. 1987.

[5] F. Highland, C. Iwaskiw. Knowledge Base Compilation. Proceedings IJCAI '89.

[6] G. Abadía, P. Lindenfelzer. Data Acquisition for Real-Time Expert Systems. Gensym Corp. Proceedings CNIASE '90.

[7] J. Debenham. The Implementation of Expert, Knowledge-Based Systems. Proceedings IJCAI '89.

[8] J. Ramírez. Las Redes Neurales y el Control de Procesos (I). Computerworld de Venezuela. 1989.

[9] J. Ramírez. Las Redes Neurales y el Control de Procesos (II). Computerworld de Venezuela. 1989.

[10] J. Stankovic, W. Halang, M. Tokoro. Real-Time Systems. Kluwer Academic Publishers. 1990.

[11] K. Ogata. Ingeniería de Control Moderna. Editorial Dossat, S.A. 1979.

[12] L. Becker et al. Using Manufacturing Process Representations. AI EDAM. 1989.

[13] R. Dodhiawala, N. Sridharan, P. Raulefs, C. Pickering. Real-Time AI Systems: A Definition and An Architecture. Proceedings IJCAI '89.

[14] R. Moore. Real-Time Expert Systems in the Information Age. AI EXPERT. 1987.

[15] R. Washington, B. Hayes-Roth. Input Data Management in Real-Time AI Systems. Proceedings IJCAI '89.

[16] S. Shekhar, S. Dutta. Minimizing Response Times in Real Time Planning and Search. Proceedings IJCAI '89.

[17] W. Perkins, A. Austin. Adding Temporal Reasoning to Expert-Systems-Building Environments. IEEE. Knowledge and Data Engineering. 1990.

INTEGRACION DE MODELOS DE DESEMPEÑO EN UNA METODOLOGIA DE DESARROLLO DE SISTEMAS

Alberto VALDERRUTEN - Yves RAYNAUD

Institut de Recherche en Informatique de Toulouse
Equipe SIERA - Université Paul Sabatier
118, Route de Narbonne 31062 Toulouse Cedex - France
e-mail: valderru@irit.fr

Resumen

Cuando la especificación de un sistema incluye restricciones de desempeño como los tiempos de respuesta esperados, su diseño debe realizarse siguiendo criterios cuantitativos adicionales, que se toman en cuenta mediante Modelos de Desempeño.

En este artículo situamos la integración del proceso de Evaluación del Desempeño en el desarrollo de un sistema, vista en términos de coexistencia entre Modelos de Diseño. Los Modelos de Desempeño deben reflejar la información contenida en otros modelos; con este propósito, identificamos las interacciones entre la Metodología de Desarrollo del sistema y el proceso de Evaluación del Desempeño.

Como ilustración, derivamos un Modelo de Redes de Colas a partir de un modelo construido con la Técnica de Descripción Formal LOTOS. Este trabajo fue realizado en el marco del proyecto COMPLEMENT, cuyo propósito es el estudio, adecuación e integración de métodos y herramientas para el desarrollo de Sistemas Tiempo Real.*

Palabras clave: Modelos de Desempeño, Modelos de Diseño, Proceso de Evaluación del Desempeño, Metodología de Desarrollo de Sistemas, Técnicas de Descripción Formal, Redes de Colas.

* *COMPLEMENT es un proyecto financiado parcialmente por la CEE en el programa ESPRIT II (proyecto No. 5409).*

1. Objetivos

Las técnicas de Evaluación del Desempeño pueden darnos información sobre el comportamiento de un sistema visto en términos de tiempo, y así estimar la calidad del servicio ofrecido (tiempos de respuesta) al igual que su eficacia (utilización correcta de los recursos). Estas técnicas pueden clasificarse según utilicen o no alguna forma de abstracción (modelos) que permitan deducir los valores para los diferentes Criterios de Desempeño. Entre las técnicas de modelado para la Evaluación del Desempeño más populares, encontramos las Redes de Colas [4, 5, 7], que permiten expresar los fenómenos de contención que se presentan en un Sistema Informático.

Realizar una Evaluación del Desempeño al mismo tiempo que el sistema es desarrollado implica tener en cuenta su comportamiento como criterio de diseño, lo cual es determinante en aquellos sistemas de reacción rápida (Tiempo Real, Sistemas Embarcados) y cada vez más exigido en sistemas de tipo conversacional. Sin embargo, los métodos de desarrollo tradicionales no permiten la expresión de exigencias de desempeño en términos de tiempo, ya que enfocan principalmente los "aspectos funcionales" (el sistema debe realizar una serie de funciones) dejando a un segundo plano los "aspectos no funcionales", que son tomados en cuenta una vez el sistema realizado. Los inconvenientes de esta práctica que son ámpliamente comentados en [10].

Nuestro objetivo es incluir la Evaluación del Desempeño en el desarrollo del sistema, apoyándonos en técnicas de modelado, lo que supone que los modelos obtenidos deben integrarse en la metodología de desarrollo utilizada. Un análisis del papel que pueden cumplir estos modelos en las diferentes etapas del ciclo de vida del sistema fue ya presentado en [1]. El siguiente paso es estudiar la complementaridad entre los Modelos de Desempeño que pretendemos introducir y los que se utilizan para diseñar el sistema en el marco de una metodología de desarrollo.

2. Integración de Modelos

El Diseño de un sistema se realiza generalmente mediante un proceso iterativo: una hipótesis es formulada, representada con la ayuda de algún formalismo (es decir, modelada), evaluada según criterios de eficacia y de viabilidad, y si la evaluación no satisface entonces se revisa la hipótesis.

Los estudios de modelado para la Evaluación del Desempeño siguen un proceso análogo: una técnica de modelado del desempeño es el formalismo de representación (las Redes de Colas, por ejemplo), y los criterios de evaluación adicionales son las exigencias cuantitativas de desempeño.

Un conjunto de modelos puede así coexistir dentro de un mismo proceso de diseño, cada uno de ellos poniendo de relieve un aspecto específico del sistema. Una relación existe entre el grado de abstracción de cada modelo, la técnica que lo engendra y los criterios de evaluación que lo motivan. El método de desarrollo se define como el conjunto de actividades que conduce a la obtención de

uno o varios modelos utilizando una o varias técnicas. Lo que nos proponemos es estudiar la manera de integrar una nueva técnica (en este caso, de modelado del desempeño) a un método de desarrollo.

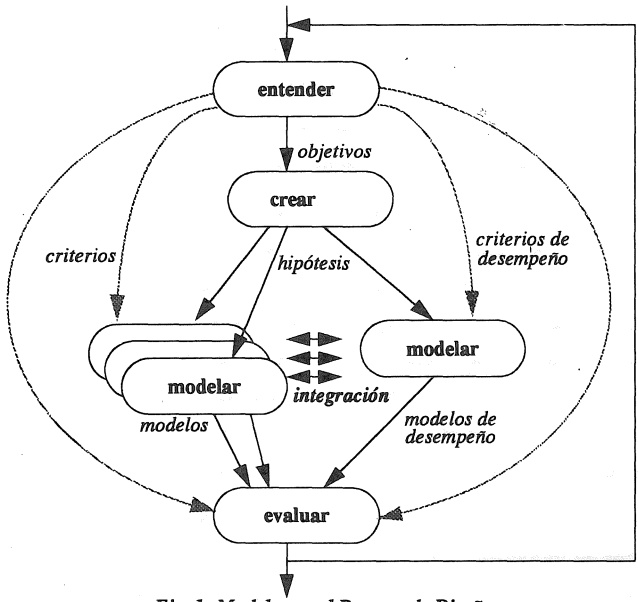


Fig. 1: Modelos en el Proceso de Diseño

La figura 1 sitúa la problemática de integración de un proceso de Evaluación del Desempeño en términos de coexistencia entre varios modelos. Aunque cada uno de ellos presente el diseño del sistema bajo un punto de vista diferente, ciertos conceptos se pueden encontrar repetidos en varios modelos. La idea es facilitar la creación de Modelos de Desempeño, utilizando al máximo la información procedente de otros modelos.

3. Puntos de Interacción con el Desarrollo del Sistema

El estudio de las interacciones entre el proceso de modelado del Desempeño y el proceso de desarrollo del sistema nos conduce a la definición de un conjunto de puntos de interacción. Cada uno está definido mediante la información que se intercambia en ellos: podemos detectar así el instante apropiado para emprender un estudio de Evaluación del Desempeño durante el desarrollo del sistema.

La interacción entre el proceso de Evaluación del Desempeño y el Desarrollo del Sistema tiene lugar básicamente en cinco puntos:

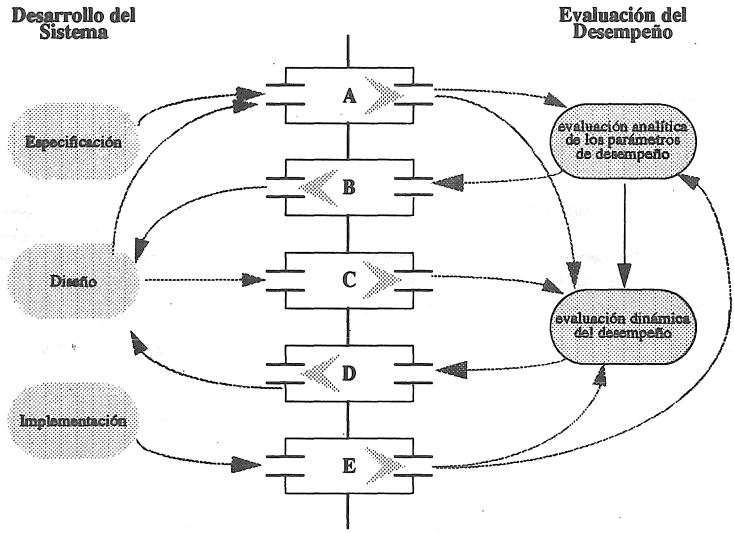


Fig. 2: Puntos de Interacción con el Desarrollo del Sistema

Los puntos A, C y E son entradas para el proceso de Evaluación del Desempeño. La información correspondiente está constituída por:

- las exigencias de desempeño en términos de tiempo
- las hipótesis de diseño consideradas
- un conjunto de factores de comportamiento, que pueden ser medidos sobre otros sistemas, calculados a partir de las características conocidas, o sencillamente estimados

Los puntos B y D corresponden a los resultados del proceso de Evaluación del Desempeño, que incluyen:

- las estimaciones obtenidas a partir de los modelos sobre el comportamiento general del sistema
- las alternativas de diseño que se puedan identificar en el caso de que algún problema en el comportamiento fuera detectado

La Tabla 1 detalla la información implicada en los puntos de interacción.

4. El proceso de Evaluación del Desempeño

Un modelo del proceso de Evaluación del Desempeño fue presentado en [3]. Su contenido no se detallará en este artículo. Sin embargo, al estudiar su interacción con el desarrollo del sistema se destacan dos etapas de este proceso (Fig. 2):

- una evaluación analítica de los parámetros de desempeño, necesaria para la construcción de los modelos [10], que además permite realizar un análisis "optimista" del comportamiento del sistema,

donde los problemas de contención no son aún tomados en cuenta;

- una evaluación dinámica del comportamiento del sistema, resultado de la utilización de los Modelos de Desempeño.

El resultado de este proceso es la evaluación del Diseño con los criterios de desempeño.

entrada A	**salida B**	**entrada C**	**salida D**	**entrada E**
exigencias de desempeño (medidas especificadas)	frecuencias de solicitación de los componentes	configuración de los recursos (topología)	*throughput* (cantidad de trabajo útil)	escenarios representativos de la carga
escenarios representativos de la carga	tiempos de servicio de los componentes	parámetros de configuración (talla de memorias...)	utilización de los componentes	diagramas de ejecución (incluyen probabilidades)
talla de los componentes del *software*	consumo de recursos / componente	parámetros de carga complementarios	tiempos de respuesta	factores de desempeño del *hardware*
diagramas de ejecución (incluyen probabilidades)	consumo total de recursos / escenario	políticas de gestion de los recursos	ocupación media del sistema	utilización de los recursos por los componentes
factores de desempeño del *hardware*	tiempos de respuesta "optimistas"	factores de gestion (*overheads*)	tiempos de espera / tiempos de acceso	parámetros de carga complementarios
factores de desempeño del entorno operativo	utilización media de los recursos *hardware*	alternativas de diseño	capacidad del sistema	políticas de gestion de los recursos
utilización de los recursos por los componentes	alternativas de diseño		recursos críticos (*bottlenecks*)	factores de gestion (*overheads*)
			alternativas de diseño / configuración	alternativas de configuración

Tabla 1: Información de desempeño

Frente a las diferentes etapas del desarrollo del sistema, identificamos qué tipo de información debe utilizarse en el modelado del desempeño. Esta información proviene de otros modelos de desarrollo.

De las Especificaciones se extraen (entrada A) las restricciones de desempeño en términos de tiempo (generalmente se especifican los tiempos de respuesta esperados). Estas exigencias sólo pueden ser consideradas de manera efectiva si son lo suficientemente precisas [12, 13], y están acompañadas de escenarios de utilización del sistema, lo que supone un trabajo importante a nivel del análisis de las necesidades.

Del Diseño extraemos (entradas A y C) la información correspondiente a las estructuras de

ejecución de los escenarios, las características del ambiente de implantación considerado, la utilización de sus recursos y las alternativas de diseño que se quieran comparar.

Desde las primeras etapas de la Implementación podemos obtener una serie de valores medidos sobre el sistema (entrada E), que reemplazan a las estimaciones hechas para la construcción de los modelos. El resultado es una herramienta para la validación del sistema, al mismo tiempo que se mejoran las estimaciones para proyectos futuros. Estas ventajas dependen de un esfuerzo de instrumentación del sistema con monitores, cuyo costo disminuye sensiblemente si su realización está prevista durante el Diseño del sistema. No nos extenderemos más con otras aplicaciones que pueden tener los monitores, pero evidentemente la validación de Modelos de Desempeño no es la única actividad favorecida con su implementación.

5. Aplicación

El estudio de integración de modelos que presentamos a continuación fue realizado en el marco del proyecto COMPLEMENT, en el cual se realiza un panorama de técnicas, métodos y herramientas para el desarrollo de Sistemas Tiempo Real, con el objetivo de proponer soluciones de integración completas y viables.

Presentamos un ejemplo de derivación de un Modelo de Desempeño a partir de un modelo de diseño. Para evitar ambigüedades en la descripción de los modelos de diseño, preferimos partir de una Técnica de Descripción Formal estándar como LOTOS [14]. El uso de este tipo de técnicas, en constante progresión, se justifica por sus capacidades de Verificación y Validación. Al integrar el modelado del desempeño, pretendemos añadir otra dimensión de verificación, guiada por la predicción del comportamiento.

Escogimos un problema suficientemente analizado como para tener una buena referencia para la validación de nuestros modelos: un protocolo de comunicación tipo *stop & wait*. La información es transmitida en un sólo sentido, en paquetes, de un emisor a un receptor. Se supone un canal de comunicación que puede alterar o perder los paquetes en ambos sentidos, así que el emisor espera la confirmación que el receptor envía al recibir un paquete correcto antes de emitir un nuevo paquete. En caso de error, el receptor no responde. El emisor gestiona con un mecanismo de *time-out* la reemisión de los paquetes erróneos o perdidos.

El modelo de comportamiento del protocolo especificado con LOTOS se presenta en la figura 3. En él, dos procesos paralelos *transmitter* y *receiver* se sincronizan con un proceso *medium* a través de dos puntos de sincronización *send* y *rec*. La alteración de paquetes se asimila a su pérdida en el *medium*. El operador [] expresa el indeterminismo en el comportamiento de un proceso. Para una descripción completa de la semántica de LOTOS, nos referimos a [14].

```
specification stop&wait [get, give]: noexit
behaviour
hide send, rec in
        (transmitter [get, send, rec] ||| receiver [give, send, rec])
        |[send, rec]|
        medium [send, rec])
where
process transmitter [get, send, rec]: noexit :=
        get; send!info; sending [get, send, rec]
where
        process sending [get, send, rec]: noexit :=
        rec !ack; transmitter [get, send, rec]
        [] i; send !info; sending [get, send, rec] (* timeout *)
        endproc (* sending *)
endproc (* transmitter *)
process receiver [give, send, rec]: noexit :=
        rec !info; give; send!ack; receiver [give, send, rec]
endproc (* receiver *)
process medium [send, rec]: noexit :=
        send!info;
        (       (i ; rec!info; medium [send, rec])
        []      (i ; medium [send, rec]))
[]      send!ack;
        (       (i ; rec !ack; medium [send, rec])
        []      (i ; medium [send, rec]))
endproc (* medium *)
endspec (* stop&wait *)
```

Fig. 3: Modelo LOTOS del protocolo stop & wait

El objetivo de una Evaluación del Desempeño es en este caso la predicción del *Throughput* del canal de comunicación. Rico y Bochmann presentan los resultados obtenidos con diferentes técnicas de modelado del desempeño [9], y los comparan a los que obtienen al utilizar una variante de LOTOS (Stochastic LOTOS) que introduce el factor tiempo. A diferencia de este procedimiento que modifica el método de diseño, en este caso LOTOS, optamos por hacer uso de su misma semántica para derivar un Modelo de Desempeño, utilizando las Redes de Colas como técnica y el lenguaje de modelado QNAP2 [8] como herramienta.

El modelo de Redes de Colas del protocolo *stop & wait* de la figura 4 se obtuvo a partir de su modelo LOTOS y de las mismas especificaciones de desempeño utilizadas en [6, 9]:

- escenario: el emisor siempre tiene paquetes por enviar al receptor
- el canal de comunicación es de 9600 bps
- paquetes de 1024 bits (en ambos sentidos), tratados en cada extremidad durante 13.5 ms
- el *overhead* de emisión es de 1 ms
- la probabilidad de pérdida de un paquete es del 5%
- *timeout* de 1 s para la reemisión de un paquete

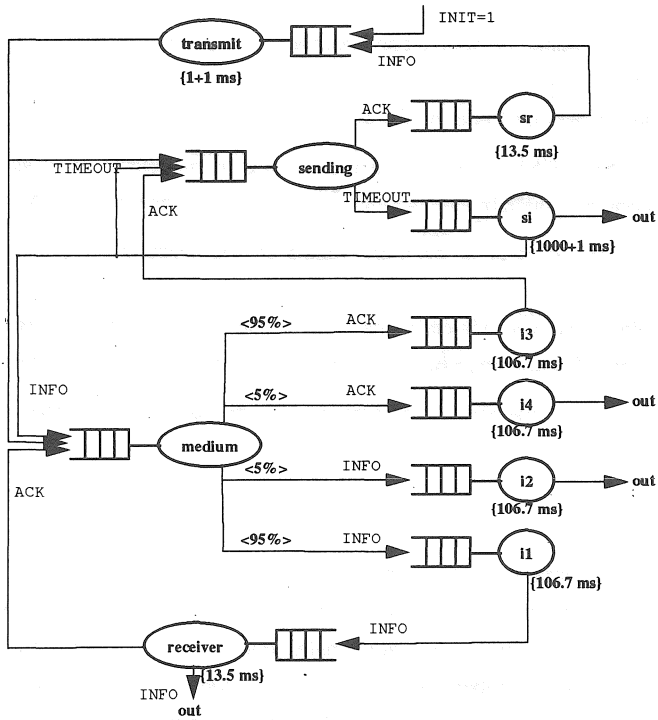


Fig. 4: Modelo de Redes de Colas del protocolo stop & wait

La simulación de este modelo produjo resultados muy próximos a los de referencia (Tabla 2). Cabe señalar que no se aplicaron los métodos analíticos de Redes de Colas (exactos o aproximados) para la resolución del modelo, ya que los mecanismos de modelado utilizados no están cubiertos por la teoría de Redes de Colas.

distribución de los tiempos de servicio	resultados obtenidos con el modelo derivado	resultado de [9] con Stochastic LOTOS	resultado de [6] con Stochastic Petri Nets
Uniforme	2.855	2.852	
Exponencial	2.863		2.75

Tabla 2: Throughput en paquetes por segundo

Con la construcción de este Modelo de Desempeño deducimos una primera serie de reglas de transformación de modelos LOTOS en modelos de Redes de Colas [2, 11]. El esfuerzo de modelado del desempeño puede verse reducido considerablemente, y limitarse a la introducción de la información relativa a las especificaciones de desempeño en un modelo de diseño.

Para tomar en cuenta estas especificaciones al mismo tiempo que se construye el modelo LOTOS, proponemos comentar el modelo con información de desempeño que se sitúe al mismo nivel de abstracción. La derivación del Modelo de Desempeño incluye la transformación de los comentarios en parámetros para las Redes de Colas.

Identificamos dónde deben situarse los comentarios en el modelo LOTOS, asi como qué información deben contener; inicialmente, adoptamos la solución consistente en añadir:

- los tiempos de ejecución de las acciones: cada acción se acompaña de **{t}**, el tiempo necesario para su ejecución.
- la probabilidad relativa a cada encadenamiento de acciones: el indeterminismo se expresa con la notación $<w_1>B_1[]...[]<w_n>B_n$, donde w_i es la frecuencia relativa del comportamiento B_i.

Esta información adicional permitió parametrar el modelo de Redes de Colas con los valores que aparecen en la figura 4.

6. Conclusión

El desarrollo de sistemas de reacción rápida (Tiempo Real, Sistemas Embarcados) o de tipo conversacional que incluyan fuertes exigencias en términos de tiempo (tiempos de respuesta) debe realizarse siguiendo criterios cuantitativos adicionales, que se toman en cuenta mediante modelos de Desempeño. Estos modelos deben coexistir con otros modelos de diseño, bien sea porque no cubren la totalidad de los aspectos funcionales del diseño, o porque otros criterios no funcionales (seguridad, robustez...) se consideran igualmente importantes.

Teniendo en cuenta este hecho, identificamos una serie de puntos de interacción entre el proceso de desarrollo del sistema (y sus modelos de diseño) y el de Evaluación del Desempeño (y sus modelos de comportamiento). Nos interesamos en la información necesaria para construir Modelos de Desempeño, identificando aquella que podemos extraer de otros modelos de diseño.

El proceso de modelado para la Evaluación del Desempeño puede facilitarse si se crean los mecanismos necesarios a la reutilización de los aspectos funcionales del diseño que aparecen en los modelos normalmente utilizados para el desarrollo del sistema.

Estudiamos la posibilidad de derivar un modelo de Redes de Colas a partir de un modelo realizado con el formalismo de LOTOS, en el cual añadimos la información sobre los tiempos de ejecución de las acciones y sus probabilidades de encadenamiento como simples comentarios que son utilizados para la derivación. Aplicamos este proceso en el caso del protocolo *stop & wait*, validamos el modelo de Redes de Colas obtenido, y deducimos un primer conjunto de reglas de transformación.

Estas reglas deberán permitir la automatización del proceso. De esta manera, el esfuerzo de modelado del desempeño se concentra en la introducción de los factores de comportamiento, que pueden ser medidos, calculados o estimados. Pero antes de llegar a una derivación automática de modelos de Redes de Colas a partir de LOTOS, dos actividades tienen que ser completadas:

- Validar el conjunto de reglas de derivación con el estudio de otro tipo de aplicaciones, verificando que se cubren los mecanismos de modelado más comunes;

- Estudiar la relación entre los factores de comportamiento, con el fin de establecer el conjunto mínimo de información adicional que debe incluirse en los modelos de diseño para cubrir las exigencias de desempeño.

7. Bibliografía

[1] J. Aguilar, A. Valderruten, *La Evaluación de Desempeño en una Metodología de Desarrollo de Sistemas de Información Distribuídos*, XVII Conf. Latinoamericana PANEL91, Caracas, Julio 8-12, 1991.

[2] A. Benzekri, D. Gazal, O. Hjiej, A. Valderruten, *LOTOS / Queueing Networks integration*, Proyecto COMPLEMENT, internal report IRI29-1.0, Enero 13 de 1992.

[3] E. Conquet, A. Valderruten, R. Trémoulet, Y. Raynaud, S. Ayache, *A Process Model of the Performance Evaluation Activity*, Summer Comp. Simulation Conf., Reno, Nevada, Julio 27-30, 1992.

[4] L. Kleinrock, *Queueing Systems*, Wiley-Interscience, 1975.

[5] S. S. Lavenberg, *Computer Performance Modeling Handbook*, Academic Press, 1983.

[6] M.K. Molloy, *Performance Analysis Using Stochastic Petri Nets*, IEEE Trans. on Computers, Vol. C-31, No. 9, Septiembre 1982.

[7] G. Pujolle, S. Fdida, *Modèles de Systèmes et de Réseaux, II - Files d'Attente*, Eyrolles, 1989.

[8] QNAP2, *Reference Manual, V4.0*, Simulog, Bull e INRIA, 1984.

[9] N. Rico, G. v. Bochmann, *Performance description and analysis for distributed systems using a variant of LOTOS*, Tenth International IFIP WG6.1 Symp. on Protocol Specification, Testing and Verification, Julio 12-15, 1990.

[10] C. U. Smith, *Performance Engineering of Software Systems*, Addison-Wesley, 1990.

[11] A. Valderruten, O. Hjiej, A. Benzekri, D. Gazal, *Deriving Queueing Networks Performance Models from Annotated LOTOS Specifications*, 6th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, Edinburgo, Escocia, Septiembre 16-18, 1992.

[12] ESA PSS-05-0 Issue 2, *ESA Software Engineering Standards*, Febrero 1991.

[13] IEEE 610.12-1990, *IEEE Standard Glossary of Software Engineering Terminology*, Diciembre 10, 1990.

[14] ISO/TC97/SC21, tech. report No. IS 8807, *LOTOS: a formal description technique based on the temporal ordering of observational behaviour*, 1987.

Fold/Unfold Transformation of Knowledge Bases under Approximate Reasoning

Gabriel Valiente

Institut d'Investigació en Intel·ligència Artificial

Centre d'Estudis Avançats de Blanes

Camí de Santa Bàrbara, 17300 Blanes, Spain

valiente@ceab.es

Abstract

Folding and unfolding are two key program transformation techniques of wide application in logic programming. We present in this paper a natural extension of folding and unfolding for knowledge bases under approximate reasoning that preserves semantics of the knowledge base if the uncertainties of the involved rules satisfy given equations. The equations both state the minimal conditions governing the folding and unfolding of uncertain rules and define the result of the fold resp. unfold operation.

Keywords: Rule-based expert systems, approximate reasoning, knowledge base transformation, folding, unfolding.

1 Introduction

Folding is a program transformation technique that in essence is the inverse of unfolding, a form of partial evaluation of logic programs. Given a logic program P containing the two

rules

$$R : B_1, \ldots, B_m, \ldots, B_n \rightarrow A_1$$

and

$$S : B_1, \ldots, B_m \rightarrow A_2$$

the folding of R and S is the new rule

$$R \uparrow S : A_2, B_{m+1}, \ldots, B_n \rightarrow A_1$$

and the replacement of rule R by rule $R \uparrow S$ preserves the semantics of logic program P as long as no rule for A_2 other than S exists in P [4].

Correctness of folding in logic programming rests upon the closed-world assumption. The requirement above that no rule for A_2 other than S exists in P is equivalent to the sentence $B_1, \ldots, B_m \leftrightarrow A_2$ being logical consequence of the completion of P [2].

Unfolding is another program transformation technique of wide application in logic programming. Given a logic program P containing the two rules

$$R : B_1, \ldots, B_m, \ldots, B_n \rightarrow A_1$$

and

$$S : C_1, \ldots, C_k \rightarrow B_m$$

the unfolding of R and S is the resolvent rule

$$R \downarrow S : B_1, \ldots, B_{m-1}, C_1, \ldots, C_k, B_{m+1}, \ldots, B_n \rightarrow A_1$$

and the replacement of rule R by rule $R \downarrow S$ preserves the semantics of logic program P, because it is only doing resolution.

Further assumptions have to be made in order to extend folding and unfolding to other representation languages. We discuss in the following section the minimal assumptions

underlying an extension of folding to deal with approximate reasoning, as found in real-world rule-based expert systems.

2 Folding Uncertain Knowledge Bases

A natural extension of folding to knowledge bases under approximate reasoning can be obtained by considering uncertain rules. In order to achieve generality, we make no assumptions regarding the nature of the certainty value L_R associated to each rule R. These values can then stand for certainty values in any particular uncertainty model used.

Given a knowledge base K containing the two rules

$$R : B_1, \ldots, B_m, \ldots, B_n \rightarrow A_1 \ (L_R)$$

and

$$S : B_1, \ldots, B_m \rightarrow A_2 \ (L_S)$$

we define the folding of R and S under approximate reasoning to be the new rule

$$R \uparrow S : A_2, B_{m+1}, \ldots, B_n \rightarrow A_1 \ (L_{R \uparrow S})$$

and want to find the conditions under which the replacement of rule R by rule $R \uparrow S$ preserves the semantics of the knowledge base K, besides the natural requirement that no rule for A_2 other than S exists in K.

It is evident that folding $R \uparrow S$ preserves the semantics of K if, for any assignment of certainty values, the certainty for A_1 obtained through R is identical to that obtained through S and $R \uparrow S$. We proceed then to compute these certainties. In the following, $\rhd$ stands for the uncertainty propagation (implication) operator and Δ stands for the conjunction operator.

The certainty for A_1 obtained through R is

$$V(A_1) = \rhd(L_R, \triangle(V(B_1), \ldots, V(B_n))).$$

On the other hand, the certainty for A_1 obtained through S and $R \uparrow S$ is

$$V'(A_1) = \rhd(L_{R\uparrow S}, \triangle(V(A_2), V(B_{m+1}), \ldots, V(B_n)))$$

where the certainty for A_2 is

$$V(A_2) = \rhd(L_S, \triangle(V(B_1), \ldots, V(B_m))).$$

It follows then that

$$V'(A_1) = \rhd(L_{R\uparrow S}, \triangle(\rhd(L_S, \triangle(V(B_1), \ldots, V(B_m))), V(B_{m+1}), \ldots, V(B_n))).$$

In order to be $V(A_1) = V'(A_1)$ it must be the case that

$$\rhd(L_R, \triangle(V(B_1), \ldots, V(B_n))) =$$

$$\rhd(L_{R\uparrow S}, \triangle(\rhd(L_S, \triangle(V(B_1), \ldots, V(B_m))), V(B_{m+1}), \ldots, V(B_n))).$$

The equation can be simplified assuming that $\rhd$ and $\triangle$ are identical, as is true for instance of the MILORD system [1, 3]. We also make use of the associativity of $\triangle$.

$$\begin{aligned}
\triangle(L_R, \triangle(V(B_1), \ldots, V(B_n))) &= \triangle(L_{R\uparrow S}, \triangle(\triangle(L_S, \triangle(V(B_1), \ldots, V(B_m))), \\
&\quad V(B_{m+1}), \ldots, V(B_n))) \\
\triangle(L_R, V(B_1), \ldots, V(B_n)) &= \triangle(L_{R\uparrow S}, L_S, V(B_1), \ldots, V(B_n)) \\
L_R &= \triangle(L_{R\uparrow S}, L_S)
\end{aligned}$$

The previous result provides the foundation for knowledge base folding under approximate reasoning. The requirement for a semantics-preserving folding $R \uparrow S$ expressed by the equation above is that $L_{R\uparrow S}$ be chosen such that $L_R = \Delta(L_{R\uparrow S}, L_S)$. As $L_{R\uparrow S}$ is, in general, not unique, we take the supremum $L_{R\uparrow S} = max\{L | \Delta(L, L_S) = L_R\}$. In those systems where the conjunction operator is taken to be the minimum function, for instance, the equation also states that folding can only be done if $L_S \geq L_R$.

3 Unfolding Uncertain Knowledge Bases

We develop here a natural extension of unfolding to knowledge bases under approximate reasoning, in the same spirit of the extension of folding made in the previous section.

Given a knowledge base K containing the two rules

$$R : B_1, \ldots, B_m, \ldots, B_n \rightarrow A_1 \ (L_R)$$

and

$$S : C_1, \ldots, C_k \rightarrow B_m \ (L_S)$$

we define the unfolding of R and S under approximate reasoning to be the new rule

$$R \downarrow S : B_1, \ldots, B_{m-1}, C_1, \ldots, C_k, B_{m+1}, \ldots, B_n \rightarrow A_1 \ (L_{R\downarrow S})$$

We want to find then the conditions under which the replacement of rule R by rule $R \downarrow S$ preserves the semantics of the knowledge base.

It is evident that unfolding $R \downarrow S$ preserves the semantics of K if, for any interpretation, the certainty for A_1 obtained through R and S is identical to that obtained through $R \downarrow S$. We proceed then to compute these certainties.

The certainty for A_1 obtained through R and S is

$$V(A_1) = \triangleright(L_R, \Delta(V(B_1), \ldots, V(B_m), \ldots, V(B_n)))$$

where the certainty for B_m is

$$V(B_m) = \triangleright(L_S, \triangle(V(C_1), \ldots, V(C_k)))$$

It follows then that

$$\begin{aligned} V(A_1) \;=\; & \triangleright(L_R, \triangle(V(B_1), \ldots, V(B_{m-1}), \\ & \triangleright(L_S, \triangle(V(C_1), \ldots, V(C_k))), \\ & V(B_{m+1}), \ldots, V(B_n))) \end{aligned}$$

On the other hand, the certainty for A_1 obtained through $R \downarrow S$ is

$$\begin{aligned} V'(A_1) \;=\; & \triangleright(L_{R\downarrow S}, \triangle(V(B_1), \ldots, V(B_{m-1}), \\ & V(C_1), \ldots, V(C_k), \\ & V(B_{m+1}), \ldots, V(B_n))) \end{aligned}$$

In order to be $V(A_1) = V'(A_1)$ it must be the case that

$$\begin{aligned} & \triangleright(L_R, \triangle(V(B_1), \ldots, V(B_{m-1}), \\ & \quad \triangleright(L_S, \triangle(V(C_1), \ldots, V(C_k))), \\ & \quad V(B_{m+1}), \ldots, V(B_n))) \;=\; \triangleright(L_{R\downarrow S}, \triangle(V(B_1), \ldots, V(B_{m-1}), \\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad V(C_1), \ldots, V(C_k), \\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad V(B_{m+1}), \ldots, V(B_n))) \end{aligned}$$

The equation can be simplified under the assumption that $\triangleright$ and $\triangle$ are identical. We also make use of the associativity, commutativity, and existence of neutral element of $\triangle$.

$$\triangle(L_R, \triangle(V(B_1), \ldots, V(B_{m-1}),$$

$$
\begin{aligned}
&\triangle(L_S, \triangle(V(C_1), \ldots, V(C_k))),\\
&\qquad V(B_{m+1}), \ldots, V(B_n))) \;=\; \triangle(L_{R\downarrow S}, \triangle(V(B_1), \ldots, V(B_{m-1}),\\
&\qquad\qquad V(C_1), \ldots, V(C_k),\\
&\qquad\qquad V(B_{m+1}), \ldots, V(B_n)))\\
&\triangle(L_R, V(B_1), \ldots, V(B_{m-1}),\\
&\qquad L_S, V(C_1), \ldots, V(C_k),\\
&\qquad V(B_{m+1}), \ldots, V(B_n)) \;=\; \triangle(L_{R\downarrow S}, V(B_1), \ldots, V(B_{m-1}),\\
&\qquad\qquad V(C_1), \ldots, V(C_k),\\
&\qquad\qquad V(B_{m+1}), \ldots, V(B_n))\\
&\triangle(L_R, L_S, V(B_1), \ldots, V(B_{m-1}),\\
&\qquad V(C_1), \ldots, V(C_k),\\
&\qquad V(B_{m+1}), \ldots, V(B_n)) \;=\; \triangle(L_{R\downarrow S}, V(B_1), \ldots, V(B_{m-1}),\\
&\qquad\qquad V(C_1), \ldots, V(C_k),\\
&\qquad\qquad V(B_{m+1}), \ldots, V(B_n))\\
&\triangle(\triangle(L_R, L_S), V(B_1), \ldots, V(B_{m-1}),\\
&\qquad V(C_1), \ldots, V(C_k),\\
&\qquad V(B_{m+1}), \ldots, V(B_n)) \;=\; \triangle(L_{R\downarrow S}, V(B_1), \ldots, V(B_{m-1}),\\
&\qquad\qquad V(C_1), \ldots, V(C_k),\\
&\qquad\qquad V(B_{m+1}), \ldots, V(B_n))\\
&\qquad\qquad \triangle(L_R, L_S) \;=\; L_{R\downarrow S}
\end{aligned}
$$

The previous result provides the foundation for knowledge base unfolding under approximate reasoning. The requirement for a semantics-preserving unfolding $R \downarrow S$ expressed by the equation above is that $L_{R\downarrow S}$ be chosen equal to $\triangle(L_R, L_S)$.

4 Conclusion

We have developed semantics-preserving folding and unfolding operations for knowledge bases under approximate reasoning. The operations are applicable whenever the uncertainties of the involved rules satisfy given equations. The equations also define the result of the folding resp. unfolding operation.

Potential applications of folding and unfolding under approximate reasoning include knowledge base verification, optimization and refinement in rule-based expert systems.

Acknowledgement

The ideas presented in this paper have evolved through many interactions with Lluís Godo, to whom we are very grateful.

References

[1] L. Godo, R. López de Mántaras, C. Sierra, and A. Verdaguer. MILORD: The Architecture and the Management of Linguistically Expressed Uncertainty. *Int. Journal of Intelligent Systems*, 4(4):471–501, 1989.

[2] C. J. Hogger. *Essentials of Logic Programming.* Oxford University Press, New York, 1990.

[3] R. López de Mántaras, J. Agustí, E. Plaza, and C. Sierra. MILORD: A Fuzzy Expert Systems Shell. In A. Kandel, editor, *Fuzzy Expert Systems*, chapter 15. CRC Press, Boca Raton, Florida, 1991.

[4] H. Tamaki and T. Sato. Unfold/Fold Transformation of Logic Programs. In *Proc. 2nd Int. Logic Programming Conference*, pages 127–138, Uppsala, Sweden, July 1984.

GAS
Gerador de Analisadores Sintáticos

Olinto José Varela Furtado
José Eduardo De Lucca
Edison Linhares de Oliveira

Laboratório de Integração de Software e Hardware (LISHA)
Departamento de Ciências Estatísticas e da Computação (CEC)
Universidade Federal de Santa Catarina (UFSC)
Florianópolis - Santa Catarina - Brasil

RESUMO

Este artigo descreve um gerador de analisadores sintáticos chamado GAS e fornece uma visão geral do ambiente no qual está inserido, detalhando algumas ferramentas de apoio. GAS gera analisadores sintáticos para várias linguagens através de diversas técnicas de análise sintática. Adicionalmente fornece documentação dos passos intermediários, possibilidade de solução interativa de conflitos e um simulador de análise sintática.

ABSTRACT

This paper describes a syntatic analyzer generator called GAS and provides a general vision of the environment which it is in and also gives some details about supporting tools. GAS generates syntatic analyzer for several languages through many syntatic analysis techniques. Furthermore, GAS provides intermediate steps documentation, possibility of conflicts interactive solution and a syntatic analyzer simulator.

KEYWORDS: Linguagens Formais, Compiladores, Compiler-Compilers

INTRODUÇÃO

O processamento de linguagens constitui uma atividade básica no desenvolvimento de software. Esta realidade motivou e continua motivando a pesquisa de técnicas que permitam a especificação formal de linguagens e a construção de seus processadores, o que tem resultado no surgimento de diversos formalismos para a especificação dos aspectos léxicos, sintáticos e semânticos de linguagens. Baseados nestes formalismos, surgiram muitas ferramentas capazes de automatizar desde a geração de algumas etapas do processo de compilação até a geração completa de um compilador.

Lex/Yacc [01] e LADE [02] são exemplos de ferramentas baseadas nestes formalismos e nas técnicas de compilação elaboradas a partir dos mesmos (ver também [09] [10] [11] [12]

[13] e [14]). Contudo, estas ferramentas não objetivam a transparência do processo de geração, omitindo os passos intermediários envolvidos e isolando o usuário. Além disso, as mesmas não oferecem flexibilidade nem na escolha do formalismo/notação de especificação (algumas até criando seus próprios formalismos) nem na escolha da técnica a ser utilizada na geração. Tais características impedem seu uso no ensino de linguagens formais e compiladores e dificulta sobremaneira o seu uso profissional, uma vez que obriga o usuário a adaptar-se à ferramenta e não ao contrário, como estabelecem os princípios da Engenharia de Software.

Objetivando o ensino adequado de Linguagens Formais e Compiladores e o desenvolvimento de processadores de linguagens foi concebido o Ambiente para Ensino e Desenvolvimento de Compiladores, o qual prima pela transparência na realização das tarefas e pela flexibilidade das ferramentas que o compõem. Dentre as ferramentas serão destacados neste artigo o GAS - Gerador de Analisadores Sintáticos, o Edigralc - Editor de Gramáticas Livres de Contexto e o Edigraf - Editor Gráfico de Autômatos Finitos.

O GAS será o objeto principal deste artigo, por ser uma ferramenta de muita importância prática e por sua abrangência teórica, envolvendo direta ou indiretamente uma gama considerável de fundamentos básicos da teoria das linguagens formais e das técnicas de compilação.

Nas seções seguintes será apresentada a descrição geral do Ambiente, destacando as ferramentas Edigralc e Edigraf e detalhando as principais características e potencialidades do GAS.

AMBIENTE PARA ENSINO E DESENVOLVIMENTO DE COMPILADORES E LINGUAGENS

O Ambiente para Ensino e Desenvolvimento de Compiladores e Linguagens vem sendo projetado e implementado no Departamento de Ciências Estatísticas e da Computação da Universidade Federal de Santa Catarina, com o objetivo de apoiar o ensino de técnicas formais de especificação de linguagens e fornecer meios para a criação de processadores de linguagens (tradutores, compiladores, editores dirigidos pela sintaxe etc.) de forma automática, prática e eficiente.

Estes objetivos estão presentes em todas as ferramentas que compõem o Ambiente, desde os editores até os geradores. O Ambiente pode ser subdividido, quanto a classe das ferramentas, em: Módulo de Linguagens Regulares (LR) e Módulo de Linguagens Livres de Contexto (LLC); e quanto à funcionalidade em Ferramentas de Edição, de Geração, de Simulação e de Manipulação/Transformação. A figura 1 apresenta todas as ferramentas e seus interrelacionamentos.

O conjunto de editores usado na especificação das linguagens regulares permite a edição nos seguintes formalismos: Autômatos Finitos-Tabela de Transições (EdiTab), Expres-

sões Regulares (ER-Edit), Gramáticas Regulares (Edigreg) e Autômatos Finitos-diagramas de transição (Edigraf). A migração entre as notações é automática, sendo que o Editab executa ainda transformações sobre autômatos finitos (determinização e minimização).

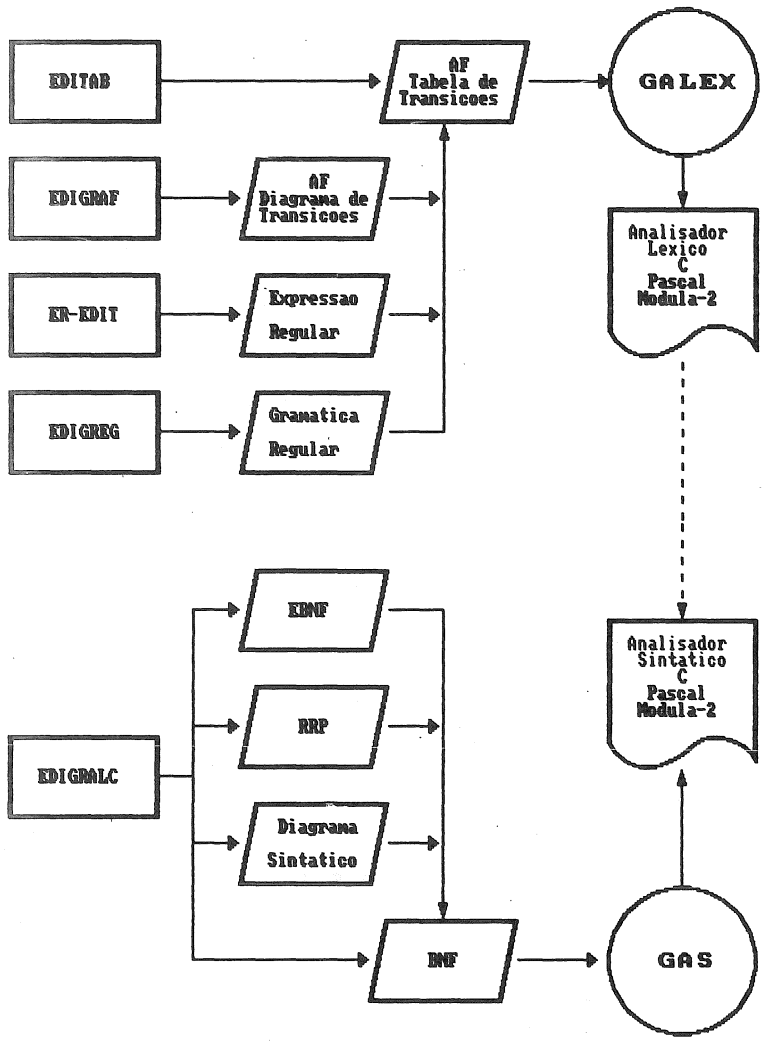


FIGURA 1 - Estrutura Geral do Ambiente

A simulação dinâmica do processo de reconhecimento léxico de uma sentença é realizado tanto pelo Editab quanto pelo Edigraf (ver adiante) e visa a depuração da especificação

e/ou compreensão do processo. O GALEX - Gerador de Analisadores Léxicos é responsável pela geração automática de analisadores léxicos, que podem ser empregados na construção de compiladores, scanners e outros programas que trabalham com aspectos regulares de linguagens. O GALEX é perfeitamente integrável ao GAS.

Para a especificação de linguagens livres de contexto a ferramenta Edigralc - Editor de Gramáticas Livres de Contexto permite utilizar as notações BNF, EBNF, RRP e Diagramas Sintáticos. A migração entre as notações também é automática. O GAS - Gerador de Analisadores Sintáticos é a ferramenta central deste módulo e usa como ferramentas de apoio os editores dos Módulos LR e LLC.

Pode-se utilizar as ferramentas de modo independente (stand-alone) ou em conjunto, o que implica na flexibilidade do Ambiente. Outro aspecto que também evidencia a flexibilidade é a possibilidade de escolha das ferramentas/notações de edição para as diversas etapas, o que liberta o usuário das limitações de outros sistemas que possuem o mesmo propósito. Além disso, utilizam-se notações exatamente como definidas na bibliografia, unindo definitivamente a teoria com a prática.

A característica de simulação dinâmica presente em algumas ferramentas - as principais - é também de fundamental importância nas duas óticas do Ambiente: para ensino pois permite a visualização e acompanhamento do processo de reconhecimento/análise facilitando a compreensão do mesmo; e para desenvolvimento como suporte à depuração da especificação e testes de desempenho.

FERRAMENTAS DE APOIO AO GAS

Tomando o GAS como peça central do Ambiente, pois centraliza e interliga os outros diversos componentes, são apresentadas a seguir algumas das principais ferramentas que servem de apoio ao GAS na tarefa de construção de analisadores sintáticos.

i) EDIGRALC - Editor de Gramáticas Livres de Contexto

O EDIGRALC permite a edição de especificações de Linguagens Livres de Contexto através de Gramáticas por quatro notações distintas: BNF - Backus-Naur Form, EBNF - Extended BNF, RRP- Regular Right-hand Production e Diagramas Sintáticos [04],[05].

Os quatro editores que compõem o Edigralc são dirigidos pela sintaxe da própria notação, ou seja, verifica-se em tempo de edição se a estrutura da notação escolhida está sendo respeitada, permitindo e apoiando as correções caso sejam necessárias. Também é realizada uma análise de consistência quanto à símbolos inalcançáveis e/ou inférteis. Estas facilidades trazem vantagens de produtividade para a edição e a ga-

rantia da correção do resultado, isto é, que a gramática editada estará no formato escolhido.

A notação Diagrama Sintático permite a especificação através da notação proposta por Conway [07], baseada em primitivas gráficas. A utilização de recursos gráficos favorece a compreensão da especificação efetuada bem como sua documentação.

O Edigralc está diretamente relacionado com o GAS, pois as gramáticas editadas são utilizados para a geração dos analisadores sintáticos.

ii) EDIGRAF - Editor Gráfico de Autômatos Finitos

O Edigraf se propõe a editar representações gráficas de autômatos finitos (Diagramas de Transição) e permitir a especificação de Linguagens Regulares, muito especialmente os aspectos léxicos de linguagens de programação. Tanto autômatos finitos determinísticos quanto não-determinísticos podem ser editados através desta ferramenta.

Possui três características fundamentais: consistência visual, simplicidade e objetividade; resultando uma ferramenta amigável e prática. Algumas características dos autômatos podem ser configurados dinamicamente: rotulação de estados automática, geração de tokens e um dicionário de estados (que contém informações sobre o estado - e que visa a documentação automática).

A simulação do processo de reconhecimento de uma sentença submetida pelo usuário é realizada passo-a-passo, de modo animado, permitindo ao usuário acompanhar o reconhecimento e depurar o autômato. Quando da simulação de um autômato finito não-determinístico, as situações de não-determinismo são tratadas com backtrack, que é indicado ao usuário e realizado automaticamente.

Uma Biblioteca Expansível de Conjuntos está disponível para agrupar símbolos do alfabeto, visando facilitar a manipulação de rótulos nas arestas e a própria visualização do autômato que está sendo editado.

Como resultado de uma edição, obtem-se um automato finito em um formato compatível com o exigido pelo GALEX (Gerador de Analisadores Léxicos), que se responsabiliza pela interação com o GAS.

GAS - Gerador de Analisadores Sintáticos

GAS é um gerador de analisadores sintáticos baseados em tabelas de parsing que aceita como entrada a especificação sintática da linguagem através de uma GLC na notação BNF quando usado stand-alone ou nas notações BNF, EBNF, RRP e de Diagramas Sintáticos quando usado de forma integrada ao Ambiente.

A GLC de entrada passa inicialmente por um processo de validação, envolvendo aspectos léxicos (verifica se os símbolos terminais e não-terminais utilizados estão dentro do padrão especificado), sintáticos (verifica se a entrada realmente é uma GLC na notação correta) e semânticos (verifica a existência de símbolos e produções duplamente definidos, existência de símbolos inférteis/inalcançáveis etc.).

Após validada a entrada, o GAS interage com o usuário permitindo a escolha da técnica de análise sintática a ser utilizada (preditiva LL(1), SLR(1), LALR(1) ou LR(1) - ver [04] [05] [08]) e a linguagem na qual o Parser deverá ser gerado (C, Pascal ou Modula-2). O GAS verifica ainda a adequabilidade da GLC à técnica escolhida, gerando documentação completa dos testes realizados. Caso as condições iniciais (relativas a cada técnica) estejam satisfeitas, passa-se à geração da Tabela de Parsing. Os conflitos porventura existentes (indicando a não adequação da GLC à técnica escolhida) podem então ser resolvidos de maneira interativa com o usuário e então a Tabela de Parsing pode ser gerada. Adicionalmente, o GAS gera arquivo com os conflitos existentes e seu histórico, permitindo que a GLC seja revista à luz dos problemas detectados.

Uma vez gerada a Tabela de Parsing, o GAS permite a simulação do processo de análise sintática dispondo de um editor de sentenças/formas sentenciais e permitindo o acompanhamento dinâmico do processo, através da visualização da pilha sintática e da ação a ser efetuada em cada passo.

As tabelas geradas são responsáveis por todo o processo de análise sintática, sendo a base para gerar o analisador e efetuar a simulação. Assim sendo, procurou-se projetar eficientemente as estruturas de armazenamento das mesmas, levando-se em consideração tanto a técnica de análise sintática envolvida como a linguagem na qual o Parser será gerado. Particularmente para as técnicas da família LR implementou-se técnicas de compactação das tabelas, resultando em redução consideravel no tamanho das mesmas (acima da média conseguida por outras ferramentas).

O Parser gerado é uma implementação exata da técnica escolhida (com a vantagem de a Tabela de Parsing estar armazenada de forma a otimizar o desempenho do mesmo), não necessitando nenhuma alteração para a inclusão dos analisadores léxicos e semânticos. O analisador léxico pode ser desenvolvido de forma independente, bastando que haja sincronização em relação aos tokens a serem utilizados; ou pode ser obtido pelo Gerador de Analisadores Léxicos (GALEX) do Ambiente, podendo usufruir das facilidades de edição e validação da especificação léxica, providas pelo Módulo LR. Verificações Semânticas e ações de Geração de Código podem ser anexados ao Parser gerado e serão ativadas automaticamente por este através de referências explícitas incluídas livremente no lado direito das produções. O momento exato da ativação destas ações pode também ser visualizado no processo de simulação, permitindo ao usuário a análise de seus efeitos no processo geral de tradução.

A flexibilidade do GAS na escolha da técnica de análise sintática propicia o ensino adequado destas técnicas, pois permite a comparação entre elas, envolvendo desde as características estruturais da GLC de entrada até a forma final do Parser gerado.

A documentação gerada automaticamente a cada passo do processo, serve como material complementar para estudo/fixação dos conceitos envolvidos, como base para uma análise mais profunda da GLC visando adequação à técnica escolhida e como documentação final da etapa de Análise Sintática dentro do processo de contrução de um compilador.

A simulação do processo de Análise Sintática possibilita ao estudante não só a percepção dinâmica do funcionamento da técnica escolhida facilitando o aprendizado da mesma como também proporciona uma visão global do processo de tradução dirigido pela sintaxe, explicitando a ativação tanto do analisador léxico, como do analisador semântico/gerador de código. Por outro lado, a simulação permite a experimentação da técnica mais adequada bem como a validação/depuração da especificação sintática da linguagem por parte do profissional.

CONCLUSÃO

Neste artigo foi apresentada uma visão geral do Ambiente para Ensino e Desenvolvimento de Compiladores e Linguagens a partir da descrição do GAS e de algumas outras ferramentas correlacionadas.

Atualmente o Ambiente está sendo utilizado no ensino de linguagens formais e compiladores nos programas de graduação e pós-graduação em Ciência da Computação da UFSC, e algumas de suas ferramentas já estão sendo utilizadas - de forma integrada ou independente - em projetos de pesquisa que envolvem direta ou indiretamente o processamento de linguagens [03].

Particularmente o GAS tem sido bastante procurado e usado como alternativa à ferramentas correlatas disponíveis, tanto em função do Ambiente que o suporta como também devido às suas características de transparência, flexibilidade, fidelidade aos aspectos formais, documentação final, possibilidade de simulação/depuração e abrangência.

Como evolução do GAS, a curto prazo, está prevista a implementação de técnicas de recuperação de erros e uso de gramáticas de atributos para especificações semânticas e geração de código. A premissa para estas e futuras evoluções do Ambiente prevê a manutenção dos objetivos e características que nortearam sua criação.

BIBLIOGRAFIA

[01] JOHNSON, S.C. - YACC - Yet Another Compiler Compiler, Technical Report 32, AT&T Bell Laboratories, 1978.

[02] XORIAN Technologies Pte. Ltd. - LADE Reference Manual, Singapura, 1991

[03] UGGIONI, D.; DE LUCCA, J. E.; FURTADO, O. J. F. - GENESE, Um Editor Genérico Dirigido pela Sintaxe, Anais de las 1as Jornadas de Ingenieria de Sistemas Informaticos y de Computacion, Quito-Equador, novembro 1990.

[04] AHO, A. V. e ULLMAN, J. D. - Compiler: Principles, Techniques and Tools, Addison-Wesley, 1986.

[05] TREMBLAY,J. P. e SORENSON, P. G. - The Theory and Practice of Compiler Writing, McGraw-Hill, 1985

[06] GOUGH, K. J. - Syntax Analysis and Software Tools, Addison-Wesley, 1988.

[07] CONWAY, M. E. - Design of a Separable Transition-Diagram Compiler, Comm. ACM, 6,7, 396-408, 1963

[08] HOPCROFT, J. E. e ULLMAN, J. D. - Introduction to Automata Theory, Languages and Computation - Addison-Wesley, 1979

[09] HORSPOOL, R. N. e LEVY, M. R. - MKScan: An Interactive Scanner Generator,Software - Practice and Experience, vol 17 num 6, 369-378, junho 1987.

[10] QUADROS, F. S. e PRICE, A. M. A. - GAL: A Lexical Analyser Generator From Graphical Specification, Pós-Graduação em Ciência da Computação, UFRGS, 1988

[11] GRUNE, D. e JACOBS, C. J. H. - A Programmer-friendly LL(1) Parser Generator, Software - Practice and Experience, vol 18 num 1,29-38, janeiro 1988.

[12] SPECTOR, D. - Efficient Full LR(1) Parser Generation, SIGPLAN Notices, vol 23 num 12, dezembro 1988.

[13] MILTON, D. R.; KIRCHOFF, L. W. e ROWLAND, B. R. - An ALL(1) Compiler Generator, SIGPLAN Notices, vol 14 num 8, agosto 1979.

[14] HEURING, V. P. - The Automatic Generation of Fast Lexical Analyzers, Software - Practice and Experience, vol 16 num 9, 801-808, 1986.

UN ENTORNO INTEGRADO DE PROGRAMACIÓN PARA EL LENGUAJE FUNCIONAL HOPE+

Velázquez Iturbide, J. Ángel
Belmonte Artero, Melchor
Castillo Sánchez, Juan Antonio

Dpto. de Lenguajes y Sistemas Informáticos e Ingeniería de Software
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo s/n, Boadilla del Monte, 28660 Madrid, España

RESUMEN

Se presenta un entorno integrado de programación para el lenguaje funcional Hope+. El conjunto de facilidades y herramientas proporcionadas permiten interactuar con el exterior, editar programas, evaluar expresiones y gestionar el entorno, todo ello bajo una interfaz de usuario homogénea. Las principales novedades residen en las facilidades de evaluación de expresiones: elección de la estrategia de evaluación y combinación de varios modos de evaluación.

PALABRAS CLAVE: entornos de programación, entornos integrados, interfaces de usuario, programación funcional, estrategias de evaluación, modos de evaluación.

1. INTRODUCCIÓN

La programación funcional ha conocido un gran auge en la década de los años ochenta, evolucionando desde el lenguaje Lisp, con sus diversos dialectos, hasta lenguajes puramente funcionales con evidentes mejoras sintácticas y semánticas [8]. Esto unido a la gran sencillez del paradigma funcional ha conllevado que sea un tema cada vez más frecuente de la enseñanza de la programación [13].

Sorprendentemente, los entornos de programación para lenguajes funcionales modernos suelen ser muy pobres (p.ej. ICHope, CAML Light); además es frecuente que sólo haya implementaciones para máquinas grandes. Quizá pueda encontrarse la explicación en que estos lenguajes están en fase de investigación y, por tanto, no se presta tanta atención al entorno como a la implementación eficiente del mismo lenguaje. La falta de entornos es una importante traba para la difusión de estos lenguajes, en concreto para su uso docente. Sin embargo, la sencillez sintáctica y semántica de los lenguajes funcionales permite prever que no es muy costoso construir entornos integrados de programación con grandes ayudas al programador.

En la ponencia se describe un entorno de programación para el lenguaje funcional

Hope+ [7, 11]. El entorno se ha concebido para un uso docente, aunque también puede usarse para producción a pequeña escala. A continuación se describen brevemente las características principales del entorno desde el punto de vista del usuario; para más detalles sobre la implementación, el lector debe remitirse a [4, 5, 15].

La estructura de la comunicación es la siguiente. En el apartado 2 se describe brevemente la interfaz de usuario proporcionada. Posteriormente, el apartado 3 expone las facilidades de evaluación del entorno y el capítulo 4, el resto de las facilidades. El apartado quinto compara el entorno con otros existentes. Por último, en el sexto apartado se hace un balance del entorno y se citan las mejoras previstas.

2. INTERFAZ DE USUARIO

El entorno presenta una interfaz de usuario moderna basada en menús de texto. De esta forma, se evita que el usuario deba aprender un nuevo lenguaje -el de mandatos de manejo del entorno- distinto de Hope+ y se evita la comisión de muchos errores de entrada -el usuario no debe escribir lo que desee sino simplemente seleccionarlo-. El formato adoptado de menús es el que se ha hecho prácticamente estándar en productos comerciales: estructura arborescente de menús, menú principal de barra, menús secundarios desplegables, selección de opciones con ayuda del cursor o de iniciales, etc.

En cuanto a la presentación de la pantalla, está dividida de forma que la línea superior contiene el menú principal, la inferior contiene mensajes informativos y el resto de la pantalla es área de trabajo. En la figura 1 puede verse el aspecto de la pantalla cuando el usuario quiere hacer alguna operación de alto nivel sobre una función **map**.

El menú principal agrupa las facilidades del entorno en cuatro categorías; en el apartado 3 se describen las facilidades de evaluación y en el apartado 4, el resto.

3. EVALUACIÓN DE EXPRESIONES FUNCIONALES

Un programa funcional se ejecuta evaluando cualquier expresión funcional que solamente contenga identificadores definidos de función y de constante; como resultado se obtiene otra expresión más sencilla, que se llama el valor de la expresión inicial. De manera abstracta, la evaluación de expresiones funcionales se define mediante una *semántica operacional*, que consta de un conjunto de reglas de reescritura de expresiones y una estrategia de aplicación de aquéllas.

Nuestro entorno basa su interfaz para evaluación en el uso de menús y un editor para que la interfaz de usuario sea homogénea en todo el entorno. El usuario edita primero la expresión y posteriormente selecciona la opción de evaluar. El entorno mantiene una lista de las expresiones editadas por lo que muchas veces no es necesario escribir una expresión completa, sino simplemente seleccionar o modificar una ya

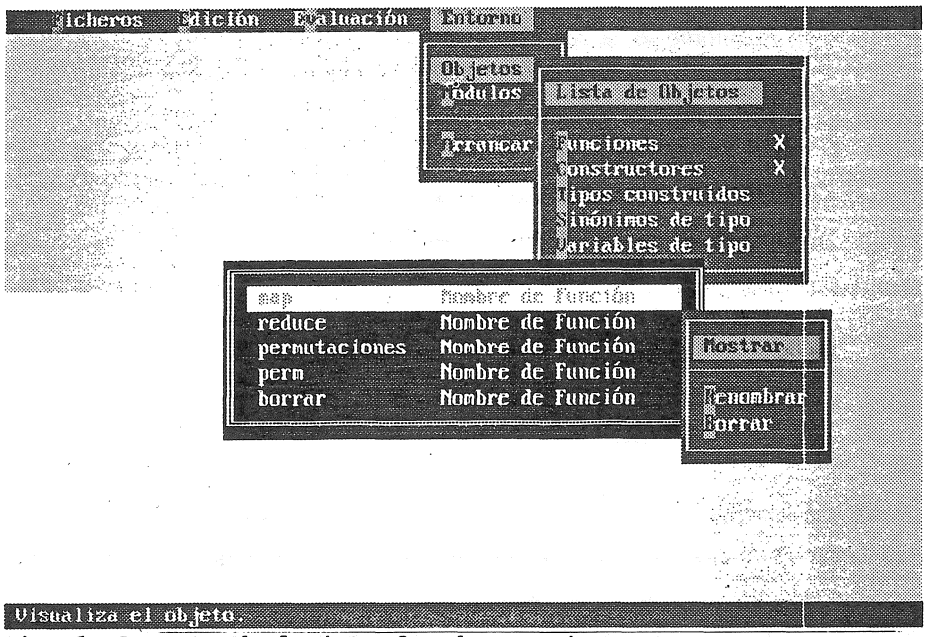


Fig. 1. Aspecto de la interfaz de usuario

existente. Obsérvese que la interacción de evaluación es distinta de la usual en intérpretes, basada en lo que se llama el "bucle de lectura, evaluación e impresión".

El entorno desarrollado presenta dos novedades. Primero, el usuario puede *elegir la estrategia de evaluación* de expresiones funcionales. Segundo, puede *combinar varios modos de evaluación* durante la misma. Veamos ambas facilidades más detalladamente.

(a) Elección de la estrategia de evaluación

Normalmente un entorno sólo permite evaluar expresiones funcionales siguiendo una estrategia prefijada por el constructor del entorno. Aunque potencialmente hay muchas estrategias de evaluación, nuestro entorno incorpora las dos más extendidas -evaluación impaciente y evaluación perezosa-. La disponibilidad de ambas estrategias permite usar la más conveniente para cada caso. Por supuesto, presenta una gran ventaja docente: los alumnos pueden experimentar y comprender mejor ambas estrategias.

(b) Combinación de varios modos de evaluación

Una vez elegida una estrategia de evaluación, el usuario puede evaluar expresiones con la ayuda del intérprete. Sin embargo, es útil disponer de herramientas que den más información que el mero valor final. En nuestro caso el usuario puede conocer expresiones intermedias que se producen durante la evaluación; estas expresiones dependen de los sucesivos modos de evaluación elegidos.

Un *modo de evaluación* es una opción que indica cuántos pasos de reescritura se van a realizar en la evaluación de una expresión, es decir, controla el progreso de la

evaluación. El entorno incluye tres modos: evaluación normal, paso a paso o avanzando hasta un punto de ruptura. El primero produce la evaluación usual en un solo paso, el segundo provoca una sola reescritura y el tercero produce la evaluación de la expresión hasta llegar a un punto de ruptura. Por último, también puede abandonarse una evaluación que, mediante las opciones segunda o tercera, está en un estado intermedio.

Una *función de punto de ruptura* es una función que provoca la interrupción de la evaluación de una expresión cuando el siguiente paso de reescritura es su aplicación a alguna subexpresión. Las funciones de punto de ruptura se seleccionan y liberan, pudiendo haber varias simultáneamente. El concepto de punto de ruptura se ha tomado de depuradores de programas imperativos y de Lisp, aunque con un fin distinto: aquí se centra la atención en la expresión que hay al ir a aplicar ciertas funciones, mientras que en los depuradores se quiere examinar el valor de variables en dicho momento.

La facilidad de evaluación paso a paso es muy útil cuando se está aprendiendo la programación funcional, sobre todo para entender la recursión y la diferencia entre las dos estrategias de evaluación. La evaluación con puntos de ruptura también es útil para comprender la recursión, así como medio de depuración.

Como facilidad adicional, una evaluación paso a paso ya concluida puede almacenarse en un fichero para su edición y examen posterior.

Veamos con un ejemplo sencillo el uso de las distintas estrategias y modos de evaluación. Supongamos la función **factorial** definida de la siguiente forma:

```
dec factorial : num -> num ;
--- factorial (n) <= if n=0 then 1 else n*factorial(n-1) ;
```

El marcado de **factorial** como función de punto de ruptura permite conocer mejor el proceso de evaluación. Eligiendo la estrategia impaciente, la evaluación de punto en punto de ruptura de **factorial(3)** produce las expresiones:

```
factorial(3)
↓
3 * factorial(2)
↓
3 * (2 * factorial(1))
↓
3 * (2 * (1 * factorial(0)))
↓
6 : num
```

La elección de una estrategia perezosa produce prácticamente las mismas expresiones salvo que la aplicación de la función se realiza sin haber evaluado previamente sus argumentos:

```
factorial(3)
↓
3 * factorial(3-1)
↓
3 * (2 * factorial(2-1))
↓
3 * (2 * (1 * factorial(1-1)))
↓
6 : num
```

Para ver otras combinaciones de modos de evaluación, veamos un último ejemplo en el que primero se avanza hasta un punto de ruptura, después se avanza paso a paso hasta la siguiente aplicación de **factorial** y ahí se termina en un solo paso. De nuevo se elige evaluación perezosa para ver mejor su comportamiento.

```
factorial(3)
↓
3 * factorial(3-1)
↓
3 * (if 3-1=0 then 1 else (3-1) * factorial((3-1)-1))
↓
3 * (if 2=0 then 1 else 2 * factorial(2-1))
↓
3 * (if false then 1 else 2 * factorial(2-1))
↓
3 * (2 * factorial(2-1))
↓
6 : num
```

4. OTRAS FACILIDADES

El entorno incluye otras facilidades imprescindibles para un entorno integrado y amistoso. Las facilidades se agrupan en tres categorías distintas:

(a) Interacción con el exterior

El entorno tiene las opciones usuales de interacción con el sistema operativo: cambiar de directorio activo, consultar el directorio activo, cargar el programa contenido en un fichero, almacenar un programa en un fichero y terminar su ejecución. Para dar mayor flexibilidad, el usuario dispone de un editor de ficheros independiente del editor de programas funcionales que puede usarse para editar texto: p.ej. una evaluación almacenada.

(b) Edición de programas Hope+

El usuario puede editar partes de un programa funcional. Para permitir mayor flexibilidad, hay dos modalidades de edición. Por una parte puede editarse el programa completo, pero también puede editarse independientemente una declaración (variable de tipo, tipo de datos nuevo o renombrado, función o uso de un módulo). El editor es de pantalla, al igual que el editor de ficheros y el de expresiones funcionales. Al terminar una sesión de edición, el programa escrito se analiza y sólo se permite la salida del editor si aquél es correcto sintácticamente o si quiere abandonarse lo editado.

(c) Gestión del entorno

Un último conjunto de facilidades necesarias son aquéllas que permiten al usuario manejar fácilmente las declaraciones incluidas. Así, puede consultarse la relación de identificadores del programa por categorías sintácticas (variables de tipo, tipos nuevos, tipos renombrados, constructores y funciones) y, tras seleccionar uno, consultarse su declaración, cambiar de identificador o borrar la declaración. También puede realizarse la misma gestión (excepto el borrado) sobre módulos cargados en el entorno. Por último, es posible borrar todas las declaraciones realizadas hasta el momento.

5. TRABAJOS RELACIONADOS

Existe una amplia oferta de entornos muy buenos dedicados al lenguaje funcional más extendido, Lisp. Nuestro objetivo actual no es superarles, sino emular sus prestaciones, aprovechándose de las ventajas que dan los lenguajes funcionales modernos, p.ej. una semántica operacional clara. El lector interesado puede consultar la relación de artículos sobre entornos de Lisp contenidos en [1, 3].

Actualmente conocemos dos entornos dedicados a Hope. El primero, llamado ICHope, es un pequeño entorno para ordenadores personales, distribuido como suplemento de un número de la revista *Byte* [2] dedicado a la programación declarativa. El entorno está basado en el bucle de lectura, evaluación e impresión y, por tanto, incluye un lenguaje de mandatos. Aunque es fácil de utilizar, y por tanto adecuado para el propósito divulgador con el que se distribuyó, tiene graves inconvenientes para desarrollos grandes.

La única facilidad de edición es la que proporcione la línea de mandatos del sistema operativo; por tanto puede ser casi inmanejable. También presenta algunos problemas el analizador sintáctico y el sistema de recogida de basura.

En [10] se comenta otro entorno, que contiene facilidades de manejo de ficheros, edición y evaluación. Pueden editarse programas en modo de texto o dirigido por sintaxis, admite declarar operadores disfijos e incluye un sistema de transformación de programas con Hope como metalenguaje y lenguaje objetivo. El artículo menciona ciertos instrumentos de análisis de programas y de rastreo de aplicación de funciones, pero no deja claro si están incluidos en el entorno. El entorno está implementado en el mismo lenguaje Hope, lo que limita su eficiencia y excluye su uso en ordenadores personales; sin embargo, demuestra que la programación funcional es un instrumento válido para desarrollar programas no triviales. Sólo contempla la evaluación perezosa para realizar entrada/salida.

Sólo conocemos otro entorno de lenguajes funcionales modernos para ordenador personal. Se trata de una implementación de un subconjunto, llamado CAML Light [9], del lenguaje CAML. Este entorno es aun peor que ICHope para el usuario. No se distribuye con un manual de usuario claro y tiene integradas muy pocas facilidades de evaluación. Como contrapartida, ofrece una amplia biblioteca de módulos y la posibilidad de trabajar con el intérprete o con un compilador, en cuyo caso crea ficheros ejecutables.

6. VALORACIÓN Y TRABAJOS FUTUROS

Dado que los entornos de programación no cubren muchas de las etapas del ciclo de vida, no son tan complejos como otros entornos de desarrollo de software [6]. Dentro de esta relativa sencillez, podemos analizar distintos aspectos del mismo.

Un aspecto importante de un entorno es la *integración* de las distintas herramientas dentro del entorno. Un entorno de desarrollo de software está integrado si las distintas herramientas que incorpora están integradas en presentación, datos, control y proceso [14]. En líneas generales podemos decir que estas condiciones se cumplen. Así, la integración de presentación se asegura con un uso homogéneo de una interfaz basada en menús y editores. La integración de datos se garantiza con una única representación de los programas en memoria; para garantizar una evaluación eficiente, los programas se guardan en un código intermedio en forma de grafos [4, 7, 12]. La integración de control se garantiza con una programación modular de las facilidades y herramientas del entorno. La integración de control se ha intentado conseguir mediante un buen diseño de los menús, que haga fácil el uso de las distintas herramientas del entorno por el programador.

Otro aspecto importante de un entorno es su *eficiencia*, tanto de espacio de almacenamiento como de tiempos de respuesta. En nuestro caso, éstos son relativamente largos durante los procesos de traducción e interpretación.

Hay algunas mejoras que pueden hacerse al entorno. En primer lugar, pueden eliminarse algunas fuentes identificadas de ineficiencia. En segundo lugar, la interfaz puede mejorarse mediante la definición de teclas de función y el impedimento de elegir opciones de menú que no tienen sentido en un momento dado (p.ej. almacenar si no se ha editado nada). Tercero, la edición puede mejorarse haciéndose parcialmente dirigida por sintaxis; esto permitiría hacer más fácilmente comprobaciones en tiempo de edición. Por último, hay que experimentar para ver si el estilo de interacción y el diseño de menús (la integración de proceso antes mencionada) es adecuado. Esto es especialmente importante para hacer pruebas rápidas durante la depuración.

REFERENCIAS

[1] Sección especial sobre Lisp, *Communications of the ACM*, vol. 34, nº. 9 (septiembre 1991).

[2] R. Bailey, "A Hope tutorial", *Byte*, agosto 1985, págs. 235-258.

[3] D.R. Barstow, H.E. Shrobe y E. Sandewall, *Interactive Programming Environments*, McGraw-Hill, 1984.

[4] M. Belmonte Artero, "Intérprete de Hope", proyecto fin de carrera, Facultad de Informática, Universidad Politécnica de Madrid, junio 1991.

[5] J.A. Castillo Sánchez, "Un entorno de programación para el lenguaje funcional Hope", proyecto fin de carrera, Facultad de Informática, Universidad Politécnica de Madrid, junio 1992.

[6] S.A. Dart, R.J. Ellison, P.H. Feiler, y A.N. Habermann, "Software development environments", *Computer*, vol. 20, nº. 11 (noviembre 1987), págs. 18-28.

[7] A.J. Field y P.E. Harrison, *Functional Programming*, Addison-Wesley, 1988.

[8] P. Hudak, "Conception, evolution and application of functional programming languages", *ACM Computing Surveys*, vol. 21, n.º 3 (septiembre 1989), págs. 354-411.

[9] M. Mauny, "Functional programming using CAML Light", informe técnico, INRIA, mayo 1991.

[10] I. Moor, "Realistic functional programming", en *Functional Programming: Languages, Tools and Arquitectures*, coordinado por S. Eisenbach, Ellis Horwood, 1987, págs. 94-108.

[11] N. Perry, "Hope$^+$", informe técnico IC/FPR/LANG/2.5.1/7, Dept. of Computing, Imperial College, University of London, octubre 1989.

[12] S.L. Peyton Jones, *The Implementation of Functional Programming Languages*, Prentice-Hall, 1987.

[13] Á. Sánchez Calle y J.Á. Velázquez Iturbide, "Fun, rigour and pragmatism in functional programming", *SIGCSE Bulletin*, vol. 23, n.º 3 (septiembre 1991), págs. 11-16.

[14] I. Thomas y B.A. Nejmeh, "Definitions of tool integration for environments", *IEEE*

Software, vol. 9, nº. 2 (marzo 1992), págs. 29-35.

[15] J.Á. Velázquez Iturbide, M. Belmonte Artero y J.A. Castillo Sánchez, "Arquitectura de un entorno integrado de programación funcional", *Primer Congreso Nacional de Programación Declarativa (PRODE'92)*, Madrid, 28-30 septiembre 1992, póster aceptado.

MODELO ENTIDAD RELACION EXTENDIDO + DF

Maria Esther Vidal Alonso Márquez

Universidad Simón Bolívar

Apartado 89000,Caracas 1080-A, Venezuela

sun!emsca!usb!mvidal@sun.com

sun!emsca!usb!alonso@sun.com

Resumen

En este artículo, se dará la definición formal a través del Lenguaje de la Lógica de Primer Orden y conceptos de la Teoría de Conjuntos, de algunas de las abstracciones del Modelo Entidad Relación Extendido + df. El modelo E-R-E +df es una extensión al modelo Entidad Relación [3, 6] en el cual se consideran además de los conjuntos entidad y relación, las abstracciones de Clasificación y Tificación de cualquiera de estos conjuntos de objetos. [5] Esta definición consta de un grupo de axiomas que definen la semántica de las abstracciones que ofrece el modelo y, un grupo de restricciones mediante las cuales se expresan las condiciones que se deben cumplir cada vez que una abstracción es usada. Dado que en esta especificación se define el Modelo E-R-E + df en función del Modelo E-R + df + RI, al utilizar el algoritmo de traducción [9] se pueden obtener traducciones de esquemas E-R-E +df a Relacional + RI que preservan toda la semántica.

Keywords: Modelo E-R, Modelo E-R-E + df, Abstracciones, Clasificación, Tipificación, Restricciones de Integridad.

Introducción

Tanto el modelo Entidad-Relación [3] como sus diversas extensiones, es ampliamente usado como herramienta de especificación conceptual, a pesar de que no existe una definición formal satisfactoria del mismo.

Por ejemplo considérese el siguiente esquema de la versión del modelo E-R que permite representar los límites de la cardinalidad [9]:

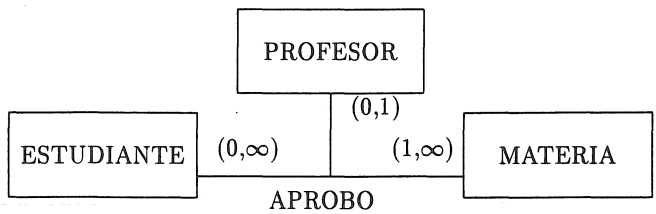


Fig. 1

El límite (0,1) de la cardinalidad del conjunto entidad $PROFESOR$ en $APROBO$, podría interpretarse de dos maneras:

1. Como una función parcial f_1 tal que,

   $f_1 : ESTUDIANTExMATERIA \longrightarrow PROFESOR.$

   Siendo ambigua la interpretación del límite inferior igual a 0.

2. Como una función f_1 total tal que,

   f_1: $EST - MAT \longrightarrow PROFESOR\ U\ \{NULO\}$, donde

   $EST - MAT = \{(est, mat)/est \in ESTUDIANTE, mat \in MATERIA,$ *est aprobo mat*$\}$

   Interpretándose el límite inferior igual a cero como que existen elementos (est, mat) que a pesar de cumplir que *est aprobo mat* no tienen asociado un elemento en $PROFESOR$.

En este artículo se presentará y se definirá formalmente el Modelo Entidad Relación Extendido con Dependencias Funcionales , y la interpretación que se le dará a la cardinalidad es esta última interpretación.

Esta especificación está dividida en un conjunto de axiomas mediante los cuales se define el significado de las abstracciones y, un conjunto de restricciones a través de las cuales, se presentan las condiciones que deben cumplirse cada vez que se utiliza una abstracción.

Así si se denota con EC_i el conjunto de fórmulas que describen el esquema ERE + df de una base de datos BD_i, $AXIOMAS(EC_i)$ el conjunto de las fórmulas lógicas que expresan el significado de cada una de las abstracciones utilizadas en EC_i, $EE(EC_i)$ el esquema por extensión de EC_i, RI las restricciones de integridad de cada una de las abstracciones utilizadas en EC_i, entonces: $\forall\ EC_i$, $AXIOMAS(EC_i)\ U\ EE(EC_i)\ U\ EC_i \models RI$, es decir, todas las estructuras de interpretación que sean modelos del conjunto de fórmulas AXIOMAS(ECi) U EE(ECi) U ECi son modelos también del conjunto RI. [1, 7].

Modelo E-R-E + df

El modelo E-R-E +df es una modificación del modelo Entidad-Relacion-Extendido presentado por Teory y Fry al cual, para aumentar su poder expresivo se han agregado entre otras cosas: conjuntos de relaciones que relacionan conjuntos de entidades y/o conjuntos de relaciones. Límites en la cardinalidad y, Tipificación y Clasificación de un conjunto entidad o relación en conjuntos de entidades o relaciones.

Los axiomas y las restricciones de integridad que cumplen algunas de las abstracciones del Modelo son:

Conjunto Relación: asociación entre conjuntos de entidades o conjuntos de relaciones. Si R es el nombre del conjunto relación que relaciona a los conjuntos de entidades o conjuntos relaciones con nombres $O_1, ..., O_n$ entonces:

$$\forall R \forall O_1 V o_1 .. \forall O_n \forall on(objeto(o_1, O_1) \& .. \& objeto(o_n, O_n) \& relacion(R(o_1, .., o_n)) \Rightarrow$$

$$nombre - conjunto - relacion(R, O_1, .., O_n)).$$

Conjunto de Tipificación: asociación entre un conjunto de entidades o relaciones llamado conjunto de objetos madre, con una lista de conjuntos de entidades o relaciones llamados conjuntos de objetos hijos. Si O es un conjunto de objetos que se tipifica en los conjuntos de objetos $O_1, .., O_n$ y a esta tificación se denomina con T, entonces se cumple el siguiente axioma:

$$\forall T \forall O \forall O_1, .., \forall O_n (nombre - conjunto - tipificacion(T, O, O_1, .., O_n) \Rightarrow$$

$$nombre - conjunto - objeto(O)\&$$

$$nombre - conjunto - objeto(O_1)\&..\&$$

$$nombre - conjunto - objeto(O_n))\&$$

$$\exists T_1..\exists T_n((nombre - conjunto - relacion(T_1, O, O_1, .., O_n)\&$$

$$cardinalidad(T_1, O_1, (0,1))\&cardinalidad(T_1, O, (1,1)))\&..\&$$

$$(nombre - conjunto - relacion(T_n, O, O_n)\&$$

$$cardinalidad(T_n, O_n, (0,1))\&$$

$$cardinalidad(T_n, O, (1,1)))).$$

Además se cumple que para cada ocurrencia del conjunto de objetos madre O existe una ocurrencia de alguno de los conjuntos de objetos hijos $O_1, .., O_n$ con el cual se tipifica a traves de T.

$$\forall T \forall O \forall O_1..\forall O_n (nombre - conjunto - tipificacion(T, O, O_1, .., O_n) \Rightarrow$$

$$\exists!o\exists!o_1..\exists!o_n(objeto(o,O)\& \quad **$$

$$((objeto(o_1,O_1)\&relacion(T_1(o,o_1))) \vee ..\vee$$

$$(objeto(o_n,O_n)\&relacion(T_n,(o,o_n)))).$$

Conjunto de Clasificación: asociación entre un conjunto de entidades o relaciones llamado conjunto de objetos madre, con una lista de conjuntos de entidades o relaciones llamados conjuntos de objetos hijos dependiendo del valor de un grupo de atributos del conjunto de objetos madre. Si O es un conjunto de objetos que se clasifica en los conjuntos de objetos $O_1,..,O_n$ segun los valores $V_1,..,V_n$ que tomen la lista de atributos L de O y, a esta clasificación se denomina con C, entonces se cumple el siguiente axioma:

$$\forall T\forall O\forall O_1..\forall O_n\forall L\forall V_1...\forall V_n(n-cjto-clasif(T,O,L,(V_1,O_1,..,(V_n,O_n)) \Rightarrow$$

$$nombre-conjunto-objeto(O)\&$$

$$nombre-conjunto-objeto(O_1)\&..\&$$

$$nombre-conjunto-objeto(O_n))\&$$

$$\exists T_1..\exists T_n((nombre-conjunto-relacion(T_1,O,O_1,..,O_n)\&$$

$$cardinalidad(T_1,O_1,(0,1))\&cardinalidad(T_1,O,(1,1)))\&..\&$$

$$(nombre-conjunto-relacion(T_n,O,O_n)\&$$

$$cardinalidad(T_n,O_n,(0,1))\&$$

$$cardinalidad(T_n,O,(1,1)))).$$

Además siempre se cumple que para cada ocurrencia del conjunto de objetos madre O

[1] $\exists!$ se interpreta el existencial único

existe exactamente una ocurrencia de alguno de los conjuntos de objetos hijos $O_1, .., O_n$ en la cual se clasifica dependiendo del valor que tome en los atributos de L.

$$\forall C \forall O \forall L \forall V_1 \forall O_1 .. \forall V_n \forall O_n (n - cjnto - clasif(T, O, L, (V_1, O_1), .., (V_n, O_n)) \Rightarrow$$

$$\exists o \exists o_1 .. \exists o_n (objeto(o, O) \&$$

$$((objeto(o_1, O_1) \& relacion(C_1(o, o_1)) \& valor(o, L) = V_1) \nabla .. \nabla \quad **$$

$$(objeto(o_n, O_n) \& relacion(C_n, (o, o_n)) \& valor(o, L) = V_n)).$$

Entre las restricciones de integridad que permite representar el modelo se encuentra la de cardinalidad de un conjunto de objetos O en un conjunto relación R. Así si O_n es el nombre de un conjunto de objetos, R es el nombre de un conjunto relación que relaciona a los conjuntos de objetos $O_1, .., O_n$ y la cardinalidad de O_n en R es min, max entonces $cardinalidad(R, O_n, (min, max))$ y siempre se cumple que:

Si $o_1, .., o_{n-1}$ son elementos del dominio de O_n en R, es decir, pertenece al conjunto de elementos de $O_1 x .. x O_{n-1}$ que tienen sentido dentro de R, entonces $o_1, .., o_{n-1}$ se relaciona a través de R como mínimo con min elementos de O_n y como máximo con max.

$$\forall O_1 .. \forall O_n \forall R(cardinalidad(R, O_n, (min, max)) \Leftrightarrow$$

$$\forall o_1 .. \forall o_{n-1} (objeto(o_1, O_1) \& ... \& objeto(o_{n-1}, O_n - 1) \&$$

$$nombre - dominio(RO_n, R, O_n) \& relacion(RO_n(o_1, .., o_{n-1}))$$

$$\Rightarrow$$

$$min \leq |\{(o_1, .., o_n) | relacion(R(o_1, .., o_n))\}| \leq max))$$

El dominio de un conjunto de objetos O_n en el conjunto relación R es RO_n sí RO_n es el conjunto de todos los elementos de $O_1 x .. x O_{n-1}$ que están relacionados con algún elemento

[2] ∇ se interpreta como el o exclusivo

de O_n o con el valor $NULO$. El valor $NULO$ es un elemento del cual no se tiene información alguna[8]. Un modo de justificar la introducción de esta constante, es verla como el resultado de reemplazar una variable existencial por una constante de Skolem.[2]

$$\forall RO_n \forall R \forall O_n (nombre - dominio(RO_n, R, O_n) \Leftrightarrow$$

$$\forall o_1 .. \forall o_{n-1} (relacion(RO_n(o_1, .., o_{n-1})) \Rightarrow$$

$$\exists o_n ((objeto(o_n, O_n) \& relacion(R(o_1, ..., o_n))) \vee$$

$$(nombre - conjunto - relacion(R, O_1, .., O_n UNULO)$$

$$\Rightarrow$$

$$relacion(R(o_1, ..., NULO))))).$$

Conclusiones

La especificación presentada define al Modelo E-R-E +df en función del del Modelo E-R + df más Restricciones de Integridad, lo cual trae como consecuencia que los esquemas puedan ser traducidoos sin pérdida de información.

Además, si se utiliza el algoritmo de traducción del modelo E-R + df al modelo Relacional [4, 8, 9, 10] y se agregan las restricciones existentes en el esquema E-R + df producido, se puede obtener un esquema relacional que preserva toda la semántica.

De esta forma, estos esquemas de traducción, pueden ser utilizados en una herramienta de diseño conceptual de Bases de Datos que permita realizar el diseño con el modelo E-R-E + df y produzca como salida, un esquema Relacional con algunas Restricciones de Integridad, que es la especificación recibida por la mayoría de los manejadores comerciales.

Por otro lado, esta especificación es utilizada en un manejador de Bases de Datos Lógicas de la Universidad Simón Bolívar, para implementar un manejador E-R-E + df que puede producir traducciones a Relacional.

Referencias

[1] BARRY, E. J. "Applied Database Logic" VolI Fundamental Database Issues. Prentice-Hall, 1985.

[2] CHIN-LIANG y LEE, "Symbolic Logic and Mechanical Theorem Proving" Academic Press. 1973.

[3] CHEN, P. "The Entity-Relationship Model-Toward a Unified View of Data", ACM Trans. Database Syst. 1,1, Mar 1976.

[4] CODD, E. F. " A Relational Model of Data for Large Shared Data Banks" Communications of the ACM, 13, 6, Jun 1970.

[5] SMITH y SMITH "Database Abstraccions:Aggregation and Generalization" ACM Trans. Database Syst. 2,2 June 1977.

[6] TEORY T., YANG D., FRY J. "A Logical Design Methodology for Relational Database using the Extended Entity-Relationship Model" ACM Comp. Surv. 18, 2, Jun 1986.

[7] THOMASON R., "Symbolic Logic: An Introduction". New York, Macmillan 1970.

[8] ULLMAN J. "Principles of Database and Knowledge-Base Systems" Vol I. 1988. Computer Science Press.

[9] VIDAL M. "Teoría de Normalización Modelo E-R+ df" Tesis de Maestría en Ciencias de la Computación Universidad Simón Bolívar. Caracas, Agosto 1991

[10] VIDAL M., MARQUEZ A. "Formas Normales Modelo Entidad Relación". XVII Conferencia Latinoamericana de Informática, Vol II, pag 1283, Caracas Julio 1991.

Local Nets

Daniel Yankelevich
Univ. Pisa, Dip. Informatica and HP Labs, Pisa Science Center
Pisa - ITALIA
E-mail: dany@pisa3.italy.hp.com

Abstract

Petri nets have been widely used to model and specify concurrent, distributed systems. However, they give no mean to reflect spatial distribution of resources.

In this work, an extension of P/T Petri nets is introduced in order to capture the concept of spatial distribution, specifying not only places and events but also how places are distributed in sites.

These new nets, called Local nets, are a very simple extension which is shown to be semantics respecting, and to support a methodology for the definition of systems.

Some theoretical results are given in order to establish the basis for a practical use of these nets for specification and verification, and the relation with previous existing models is also discussed.

Keywords: Petri nets, concurrent systems, specifications.

1 Introduction

Petri nets [9] have been widely used to model concurrent systems. In particular, P/T Petri nets conform a very useful tool for specification and verification of systems composed of interacting entities, since they have a nice graphical representation and present algebraic properties which allow the use of algebraic techniques for verification [11].

Petri nets have also been used to specify and analyze complex processes, not only in computer science; for example they have been used to help in the definition of big projects. They express directly the concept of *causality* and reflect the causal relationships between events. However, they do not give an idea of *distribution in space* of entities. In particular, no information on *where* an event takes place can be specified with a Petri net. Causality and distribution are strongly related, but they are not the same concept.

For example, the net of Figure 1 could be given to represent the following process:

Process One

- Two sites of a company, one in Bristol and one in Pisa, start a common development. The one in Bristol specifies one module (event sB) and that in Pisa specifies a different one (event sP).
- When both sites have finished, they exchange the specifications, in order to have a view of the whole project (event ex).
- Each site proceeds to implement its own part (events iP and iB).

In the net of Figure 1, no information on where the events take place, or of how the places are grouped together (in Pisa or in Bristol) is given. The same net specifies also the (very different) process in which Pisa implements the module that Bristol has specified and viceversa.

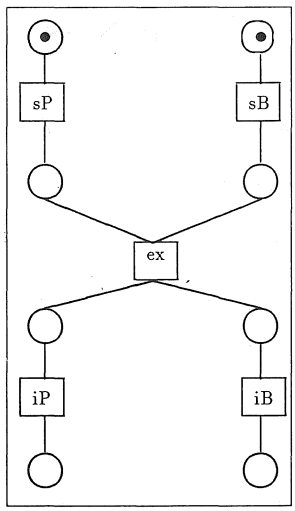


Figure 1: The net of Process One

When information about distribution is taken into account some processes that are identified using Petri nets have to be discriminated. Hence, some additional information has to be included in the specifications.

One solution could be the use of *high level* Petri nets [10]. However, for simple examples as the one given above, this seems to be an ad-hoc unsatisfactory solution: it requires to move from a propositional to a first order theory without a real need for first order features. Moreover, not all such nets have nice properties as a simple graphical representation.

On the other hand, the information about distribution should be orthogonal or independent of the underlying net, and it should not be codified inside the net in the form of colored tokens or predicates associated to events. In fact, there are three requirements for such an extension:

1. the extension should be semantics respecting, that is, if one associates some information about distribution and then (s)he forgets it, the resulting net should be equivalent to the one without extra information,

2. the extension should be orthogonal, since distribution is a different concept, and it has to allow one to specify or leave unspecified the spatial distribution, and

3. it has to be flexible enough to permit the definition of subsites of a given site (for example, as in a *fork* operation or when a team is divided to work in different subprojects), to have multiple inheritance of sites (for example, actions which involves more than one place) and sites to be destroyed or created.

Most of the existing models do not satisfy requirement (1) or (2). A simple idea would be "to group places together" in sets of places belonging to some common site, but it is easy to see that this idea do not satisfy the flexibility requirements of (3).

The problem of localities has been studied in the context of process description languages [7] by G. Boudol, I. Castellani, M. Hennesy and A. Kiehn [2]. In this work, the ideas of [2] are applied to the field of nets, defining Local nets as an extension of P/T Petri nets which satisfies the requirements stated above.

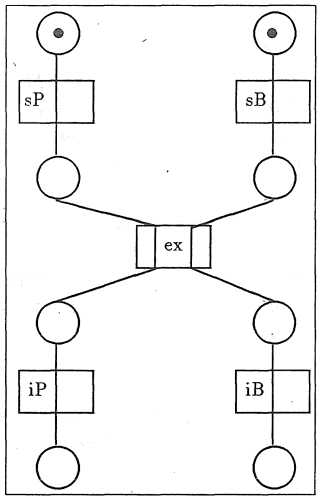


Figure 2: The local net of Process One

For example, the net of figure 1 can be extended with spatial information represented graphically as arcs inside the events to specify correctly the Process One. The extended net is shown in figure 2.

In this net, it is shown that the arc from Pisa causing the event ex continues with the line that causes the event iP. Notice that this information is not the same as having individual tokens. If the extra arcs of the net are deleted (turning transparent boxes into opaque ones), the net of figure 1 is obtained.

With this extension, the same properties of nets are still valid and the same verification mechanisms can be used. Moreover, extending existing procedures to take into account the spatial distribution is easy.

Section 2 recalls the main definitions of P/T Petri nets. Section 3 introduce local nets, and include some definitions and theorems to support the use of these nets for specification.

2 Background: P/T Petri Nets and Boolean Algebras

In this section, the main definitions of P/T nets and of lattices are recalled. For a complete presentation of Petri nets, the reader can refer to [9]. For the definitions about algebra, the reader can refer to [3].

Definition 2.1 P/T Net
A net $N = (S, T, W, M_0)$ *consists of disjoint sets* S *and* T *of places and transitions, the weight function* $W : S \times T \cup T \times S \rightarrow \mathbb{N}$ *and the initial marking* $M_0 : S \rightarrow \mathbb{N}$, *where* $\mathbb{N}$ *are the natural numbers.*

Definition 2.2 Pre and post sets
Given a net $N = (S, T, W, M_0)$ *and an element* $x \in S \cup T$,

- *the pre-set of* x *is* ${}^\bullet x = \{y \in S \cup T \mid W(y, x) \neq 0\}$

- *the post-set of x is* $x^\bullet = \{y \in S \cup T \mid W(x,y) \neq 0\}$

Definition 2.3 Markings and enabling
Given a net $N = (S,T,W,M_0)$, a marking *is a function $M : S \to \mathbb{N}$. A transition $t \in T$ is enabled under a marking M, denoted $M[t >$ if for all $s \in S$, we have $M(s) \geq W(s,t)$. An enabled transition may occur, producing a follower marking M', written $M[t > M'$, if $M[t >$ and $M'(s) = M(s) - W(s,t) + W(t,s)$ for all $s \in S$.*

The previous definition and notation is generalized for multisets of transitions G, and we write $M[G > M'$. In this case, $M[G > M'$ is called a *step*.

A marking M is called *reachable* if for some sequence of steps $G_1 \ldots G_n$ we have that $M_0[G_1 > M_1 \ldots [G_n > M$, and it is called *safe* if for all $s \in S$, it holds that $M(s) \leq 1$. A net is called safe if all its reachable markings are safe.

Definition 2.4 Processes
The tuple $\Pi = (B,E,F,p)$ is said to be a process of a given safe net N if $\Pi = (B,E,F)$ is a safe net with no initial marking and p is a function from $B \cup E$ to $S \cup T$, such that the following conditions hold:

- *the graph defined by $(B \cup E, F)$ is acyclic.*
- *for all $b \in B$ we have $|{}^\bullet b| \leq 1$ and $|b^\bullet| \leq 1$*
- *p maps elements of B to elements of S and elements of E to elements of T and the initial places[1] of Π to the places e such that $M_0(e) = 1$*
- *for each $e \in E$ and for all $s \in S$, $W(s,p(e)) = |p^{-1}(s) \cap {}^\bullet e|$ and $W(p(e),s) = |p^{-1}(s) \cap e^\bullet|$*

From the definition of a process of a safe net, it is possible to construct a partial order of events, representing the causal relations of occurrences of transitions in the process. Hence, to a *computation* of the net it is associated a partial order in the following way:

Definition 2.5 From processes to partial orders
Let $\Pi = (B,E,F,p)$ be a process of a safe net N. We associate to it the partial order $\langle E, \leq \rangle$, where $e \leq e'$ iff $(e,e') \in F^*$, where F^* denotes the reflexive and transitive closure of F.

In this order, $e \leq e'$ means that e is a necessary condition for e' to occur.

Notation 2.1 *Let $L = \langle D, \leq \rangle$ be a non-empty ordered set. We write $x \wedge y$ for the least upper bound of $\{x,y\}$ and $x \vee y$ for its greatest lower bound. Similarly, if $S \subseteq D$, we write $\bigvee S$ for its least upper bound and $\bigwedge S$ for its greatest lower bound.*

Definition 2.6 *Let $L = \langle D, \leq \rangle$ be a non-empty ordered set. The structure L is a complete lattice iff $\bigvee S$ and $\bigwedge S$ exist for all $S \subseteq D$.*

Definition 2.7 *A lattice L is said to be distributive if it satisfies:*

$$(\forall a,b,c \in L)\, a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$

Definition 2.8 *A boolean algebra is a structure $(B, \vee, \wedge, ', 0, 1)$ such that $(B, \vee, \wedge)$ is a distributive lattice, $a \vee 0 = a = a \wedge 1$, $a \in B$, $a \vee a' = 1$ and $a \wedge a' = 0$ for all $a \in B$*

[1] The initial places b of a process are the places such that ${}^\bullet b = \emptyset$.

3 Local Nets

In this section the proposed extension of Petri nets, namely Local nets, are introduced, and some results about them are presented.

Definition 3.1 Local Net
A Local net $L = (S, T, W, M_0, R)$ *is such that* (S, T, W, M_0) *is a net and* $R \subseteq (S \times T \times S)$ *is the local relation such that* $(s, t, s') \in R$ *implies* $W(s, t) \geq 1$ *and* $W(t', s') \geq 1$.

The local relation R is the relation that tells us which places are in the same location. That is, $(s, t, s') \in R$ can be interpreted as " when event t occurs s' is at the same physical place of s".

One interesting point of Local nets is that the graphical representation of Petri nets is not lost: the relation R can be drawn as arcs inside the boxes of the net.

Many examples of Petri nets specifications describe intuitively the system as composed of components: a process and a buffer, a producer and a consumer, readers, writers and an arbiter. However, from the description given in the net there is no way of recovering these agents, which play an important role in the intuitive understanding. Moreover, when the examples become more complex, it is difficult even to recognize in the graphical representation the different agents that interact in the system. From the Local net description, it is easy to individuate the different agents. In a previous work [4] the relation between automata describing individual behaviours and nets describing the whole system has been studied. The notion of local behaviours describing different subprocesses of a global system has been used previously without a formal basis. See for instance [1], where partial orders of events are classified in threads of control in order to introduce some concepts of parallel programming.

The second requirement discussed in the introduction was not to loose the properties we already have for P/T nets. In fact, it is possible to "forget" the additional structure without affecting the meaning of the net.

Definition 3.2 *Given a Local net* $L = (S, T, W, M_0, R)$, *we associate to it a net* $\mathcal{U}(L) = (S, T, W, M_0)$.

Here two immediate facts relating the behaviour of L and that of $\mathcal{U}(L)$ are stated:

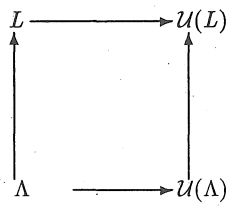
Fact 3.1 *Let* $L = (S, T, W, M_0, R)$ *be a Local net. Then,* $M_0[G_1 > M_1 \ldots [G_n > M$ *is a step sequence of* $\mathcal{U}(L)$ *if and only if* $M_0[G_1 > M_1 \ldots [G_n > M$ *is a step sequence of* L.

Fact 3.2 *Let* L *be a safe Local net. Then,* $\mathcal{U}(L)$ *is also safe.*

Processes of Petri nets are defined as a sub-class of nets without conflict and cycles. In the same way, processes of Local nets are just Local nets with the same conditions.

Definition 3.3 *A Local process* $\Lambda = (B, E, F, p, V)$ *of a safe Local net* $L = (S, T, W, M_0, R)$ *is such that* (B, E, F, V) *is a safe Local net without initial marking, and* (B, E, F, p) *is a process of* (S, T, W, M_0), *and* V *verifies that* $p(b) = s$ *and* $p(b') = s'$ *and* $p(e) = t$ *and* $(s, t, s') \in R$ *implies that* $(b, e, b') \in V$.

Theorem 3.1 *Let* L *be a safe Local net and let* Λ *be a process of* L. *Then, the following diagram commutes:*



That is, if Λ is a process of L, then $\mathcal{U}(\Lambda)$ is a process of $\mathcal{U}(L)$.

A computation of a Local net does not generate a partial order, but a richer structure.

Definition 3.4 *A colored partial order is a structure $\langle D, \leq, \sqsubseteq \rangle$ such that $\langle D, \leq \rangle$ and $\langle D, \sqsubseteq \rangle$ are partial orders and $d \sqsubseteq d'$ implies $d \leq d'$.*

Colored partial orders have been introduced in [8] as structures for giving semantics to process description languages which take into account both causal dependencies and spatial distribution. They are called colored partial orders because, since $\sqsubseteq \subseteq \leq$, it is possible to represent a finite colored partial order $\langle D, \leq, \sqsubseteq \rangle$ with the partial order diagram of $\langle D, \leq \rangle$, where the arcs belonging also to $\sqsubseteq$ are marked with a different color.

This definition permits to establish a connection between the theory of Local nets and the theory of location equivalence for process description languages. In fact, in [5] it is proved that *localities + causality = local/global causes*, which is an equivalence introduced for the CCS language [7] gluing together spatial and causal information, while in [8] it is shown that the semantics of local/global causes correspond to the semantics obtained using colored partial orders as observation domain. Hence, this characterization of computations of Local nets as colored partial orders offers a common language for computations (both of nets and of process description languages) reflecting causal and spatial information.

As a corollary of the previous lemma we can show how to obtain a colored partial order from a Local process and how to recover the causal relation from the colored partial order.

Corollary 3.1 *Let $\Lambda = (B, E, F, p, V)$ be a Local process of a Local net L. Let $C = \langle E, \leq, \sqsubseteq \rangle$ be the structure where E is the set of transitions of the process, $e \leq e'$ iff $(e, e') \in F^*$, and $e \sqsubseteq e'$ iff $(e \leq e'$ and $\exists b, b', b'' \in B.(b, e, b'), (b', e', b'') \in V)^*$, where R^* denotes the reflexive and transitive closure of the relation R. Then,*

1. *C is a colored partial order, and*
2. *the partial order $\langle E, \leq \rangle$ is the partial order associated with $\mathcal{U}(\Lambda)$.*

The following result characterize the structure of the local nets corresponding to a given P/T net.

Definition 3.5 *Let $N = (S, T, W, M_0)$ be a finite P/T net (i.e. a P/T net such that S and T are finite). Then, $\mathcal{L}_N = \{$local nets $L \,|\, \mathcal{U}(L) = N\}$ forms an ordered set with the inclusion order $(S, T, W, M_0, R) \leq (S, T, W, M_0, R')$ iff $R \subseteq R'$.*

All the local nets corresponding to a P/T net form an ordered set. In this set we have all the forms in which the spatial information can be incorporated in the net. This partial order turns out to be a complete lattice. However, lattices have few structure, which is not of much help when specifying systems. In fact, the set $\mathcal{L}_N$ forms a boolean algebra, which is a richer structure with more operations.

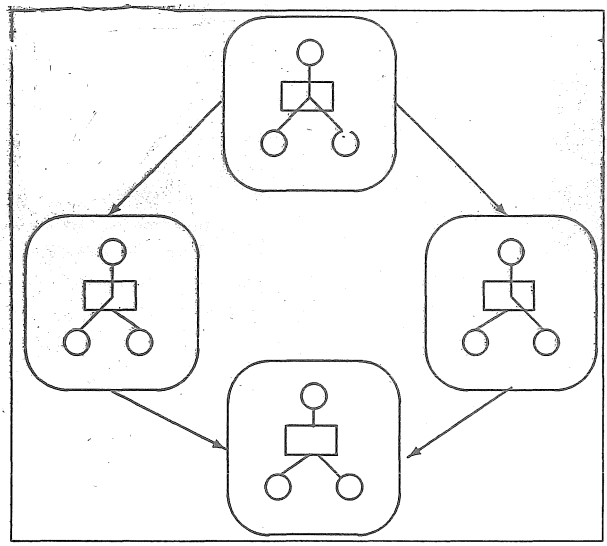


Figure 3: The boolean algebra of the *fork* operation

Lemma 3.1 *Given a finite P/T net N, the set $\mathcal{L}_N$ forms a boolean algebra.*

Proof: Let $N = (S, T, W, M_0)$. From the characterization theorem for finite boolean algebras [3, pages 164 – 165], we have that $\mathcal{L}_N$ is a boolean algebra iff it is isomorphic to the powerset of the atoms of $\mathcal{L}_N$ ordered by inclusion. It is easy to check that the set of atoms of $\mathcal{L}_N$ is the set $A = \{((s,t,s')) \in (S \times T \times S)$ such that $W(s,t) \geq 1$ and $W(t,s') \geq 1\}$ It is immediate that $\mathcal{L}_N$ is isomorphic to $\wp(A)$, and hence it is a boolean algebra. □

Example 3.1 *Let N be the Petri net which specifies a fork operation. The boolean algebra $\mathcal{L}_N$ associated to N is shown in figure 3. It has four elements, one corresponding to each spatial distribution: the bottom element leaves the spatial distribution unspecified, there are two nets for the UNIX like fork operation (where the parent process continues and one child is created) and the top specifies the division of a team in two subteams. Notice that using local nets it is possible to specify different kinds of fork operations, while with standard nets this is not possible.*

Hence, we have now some operations over local nets which suggest a natural two-step methodology in the definition of distributed systems:

1. Define a Petri net specifying the causal relationships in the system.
2. In the boolean algebra of local nets corresponding to the net of the previous step, choose one element.

With the help of lemma 3.1 the second step can be performed joining and meeting nets, taking complements, etc.

The structure proposed provides a basis for a language for spatial distribution. In fact, a very simple language (with the operations of the algebra as constructors) can be defined, and in this way a local net can be specified with a P/T Petri net and an expression of this language.

4 Conclusions and Further Work

The main result of this work is the definition of a simple extension of Petri nets (Local nets) able to specify spatial distribution without loosing interesting properties of nets and incorporating as few information as possible.

Some theoretical results are given in order to provide a basis for the use of Local nets in specifications: theorem 3.1 shows that the extension is semantics-respecting, corollary 3.1 relates this approach with a previous one in the field of process algebras [2] and lemma 3.1 gives a characterization of the structure of the local nets corresponding to a given fixed net.

The work presented here can be extended in two ways. From the definition of local processes and using the same techniques applied in the definition of colored partial orders, to define *local event structures*, extending event structures [12] with spatial information. The second extension would be to define local nets inside the algebraic context of [6].

Acknowledgments: I wish to thank Juan Vicente Echagüe for many useful comments and Cosimo Laneve because of his careful reading.

References

[1] P. Ancilotti and M. Boari. *Principi e Tecniche di Programmazione Concorrente* . Utet-Libreria, Torino, 1987.

[2] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. A theory of processes with localities. Tech. Report 13/91, University of Sussex, Brighton, 1991. To appear, Proc. CONCUR '92.

[3] B. Davey and H. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 1990.

[4] F. De Cindio, G. De Michelis, L. Pomello, and C. Simone. Superposed automata nets. In *Application and Theory of Petri Nets*, number 52 in Informatik Fachberichte, pages 229–279. Springer-Verlag, 1982.

[5] A. Kiehn. Local and global causes. Technical Report 342/23/91, Technische Universitat Munchen, Munich, 1991.

[6] J. Meseguer and U. Montanari. Petri nets are monoids. *Information and Computation*, 88:105–155, 1990.

[7] R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

[8] U. Montanari and D. Yankelevich. A parametric approach to localities, 1991. To appear, Proc. ICALP '92.

[9] W. Reisig. *Petri nets – an introduction.* EATCS Monographs on Theoretical Computer Science, Volume 4. Springer-Verlag, 1985.

[10] W. Reisig. Petri nets and algebraic specifications. *Theoretical Computer Science*, 80:1–34, 1991.

[11] W. Vogler. Deciding history preserving bisimilarity. In *Proc. ICALP '91*, volume 510 of *Lecture Notes in Computer Science*, pages 495–505. Springer-Verlag, 1991.

[12] G. Winskel. Event structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II* , volume 255 of *Lecture Notes in Computer Science*, pages 325–392. Springer-Verlag, 1987.

Compiling Timed Algebras into Timed Automata

Sergio Yovine
Laboratoire de Génie Informatique, Institut IMAG
B.P. 53X, 38041 Grenoble cedex, France
e-mail: sergio@vercors.imag.fr

Abstract

Real-time systems are hard to model, specify and design. It is common practice today to pass from informal specifications to implementations and then test the behavior of the system on some cases, which is inadequate to develop bug-free real-time systems. To ensure correctness we need to employ automatic verification methods and tools. The following real-time software development process has been proposed in [7]: ATP is used to describe real-time systems and TCTL to specify the real-time requirements. The verification approach consists in translating ATP to timed automata and then performing model-checking. In this work we present a tool that compiles ATP terms into timed automata.

Keywords: real-time systems, process algebras, temporal logics, timed automata.

1 Introduction

Typical real-time applications are control systems (e.g., flight controllers, manufacturing systems), monitoring systems (e.g., air traffic, patient monitoring) and communication systems (e.g., communication protocols). The correctness of these systems is highly dependent on the timing delays of the components. Because of the intricacies of the timing relationships, real-time systems are quite hard to model, specify and design. Consequently, there is a great demand for formal methods applicable to real-time systems.

Developing formal methods for the design and analysis of real-time systems is a very active area of current research. Several different formalisms have been proposed. These include timed Petri Nets [4], timed process algebras [9, 10, 6], real-time temporal logics [2, 5] and timed automata [1]. Moreover, there is a pressing need to develop efficient tools to be embedded in a real-time software development process.

Timed algebras are process algebras extended with a set of constructs to express timing requirements. They have formal semantics, given in terms of transition systems or sets of traces, and are a quite adequate to describe the behavior of real-time systems. ATP [6] is a timed process algebra which introduces a mechanism to delay the execution of an action and provides two language constructs, namely timeouts and watchdogs, to express timing constraints.

Timed automata have been proposed in [1, 8] to model finite-state real-time systems. Essentially, timed automata are automata extended with a finite set of real-valued variables called timers. Timing constraints are expressed by associating predicates over timers to transitions. In [8] timed automata are extended with ATP operators and a semantics is provided accordingly. In fact, this motivates an abstract formal method for translating ATP terms into timed automata.

Temporal logic has been extensively used as a formalism for specifying the behavior of concurrent systems [3]. Various real-time extensions of CTL, all called TCTL, have been proposed [1, 5]. The main result in [1] is a model-checking algorithm to check whether a timed automaton satisfies a TCTL formula. In [5] a symbolic model-checking algorithm has been presented.

The following real-time software development process has been proposed in [7]: ATP is used to describe real-time systems and TCTL to specify the real-time requirements. The verification approach consists in translating ATP to timed automata and then performing model-checking. In this work we present a tool that compiles ATP terms into timed automata.

This paper is organized as follows. In sections 2 and 3 we give a brief summary of ATP and timed automata, respectively. In Section 4 and 5 we describe the essential features of the tool.

2 ATP

In this section we define the syntax and semantics of ATP [6, 8]. We assume that a real-time system is the composition of communicating processes. A system may evolve either by *performing an action*, that is a discrete state change, or by *letting time pass*. Let Act be a set of names for actions. The communication between actions is specified by a function $\cdot|\cdot : Act_\perp \times Act_\perp \to Act_\perp$, where $\perp \notin Act$ and $Act_\perp = Act \cup \{\perp\}$. For $a_1, a_2 \in Act$, $a_1|a_2 \in Act$ is the action resulting from the communication of a_1 and a_2. If $a_1|a_2 = \perp$, then the actions cannot interact.

Time is modeled as a global state variable ranging over a *time domain* D. Some examples of time domains are $\mathbb{N}$ (discrete-time), $\mathbb{Q}^+$ and $\mathbb{R}^+$ (dense-time). An important assumption guaranteeing consistency is that time progresses synchronously for all the processes.

Definition 2.1 The syntax of the algebra is the following:

$$P ::= \texttt{idle} \mid Y \mid aP \mid \texttt{delay}(a)P \mid P + P \mid P\ \texttt{timeout}(d)\ P \mid P\ \texttt{watchdog}(d)\ P \mid P||P \mid \texttt{restrict}\ H\ \texttt{in}\ P \mid \texttt{rec}Y.P$$

where $Y \in Var$, $a \in Act$, $d \in D - \{0\}$ and $H \subseteq Act$. We consider only closed terms. ■

We informally explain the meaning of the operators of the language. A formal operational semantics is given in [6, 8].

1. `idle` denotes the process that does nothing but letting time pass.

$$
\begin{array}{rcl}
System & = & \texttt{restrict}\ data_1!, data_1?, data_2!, data_2?\ \texttt{in}\ Sens_1 || Sens_2 || Mon \\
Sens_1 & = & sample_1 \\
 & & ((\texttt{delay}(data_1!)\ \texttt{idle})\ \texttt{timeout}(2)\ data_1!\ \texttt{idle})\ \texttt{watchdog}(10)\ Sens_1 \\
Sens_2 & = & sample_2 \\
 & & ((\texttt{delay}(data_2!)\ \texttt{idle})\ \texttt{timeout}(4)\ data_2!\ \texttt{idle})\ \texttt{watchdog}(10)\ Sens_2 \\
Mon & = & \texttt{delay}(data_1?)\ \texttt{delay}(data_2?)\ Comp \\
 & & + \\
 & & \texttt{delay}(data_1?)\ \texttt{delay}(data_2?)\ Comp \\
Comp & = & \texttt{idle}\ \texttt{timeout}(1)\ result\ Mon
\end{array}
$$

Figure 1: Simple monitoring system.

2. The *prefixing* construct has two forms. aP performs the action a *immediately* and then behaves as P. In $\texttt{delay}(a)P$ the execution of a can be arbitrarily delayed.

3. The *timeout* is a very common real-time construct. Timeouts are widely used in real-time systems to safeguard one part of the system against malfunctioning of another. A timeout is generated at the end of a period of time in which a certain event has not occurred. The process $P\ \texttt{timeout}(d)\ Q$ behaves as P if P performs an action before a time d. Otherwise, it behaves as Q.

4. Sometimes, the execution of a process needs to be monitored by a *watchdog* timer. The process sets the timer and it should reset it before the timer expires. When the process does not succeed in resetting the timer in time, the process is halted by a signal from the watchdog timer. The process $P\ \texttt{watchdog}(d)\ Q$ behaves as P during a time equal to d. At time d, P is aborted and Q is started.

5. The *non-deterministic choice* $P + Q$ behaves either like P or like Q, but only with respect to actions. In the case of time, we require that $P + Q$ can wait a time d if and only if both P and Q can do so.

6. The *parallel composition* $P||Q$ allows P and Q to proceed independently and, in addition, it allows communication between them. If P can perform $a \in Act$ and Q can perform $b \in Act$, then $P||Q$ can perform the action $a|b \in Act$. Notice that P and Q cannot synchronize when $a|b = \bot$. As above, we require that in the case of time, $P||Q$ can wait a time d if and only if both P and Q can.

7. The construct $\texttt{restrict}\ H\ \texttt{in}\ P$ disallows P to perform actions that belong to H.

Example 2.1 Consider a real-time system whose purpose is to collect data from two sensors, and to compute some function of the data and then to print the result.

Each sensor takes a sample reading every 10 milliseconds and attempts to communicate the data to the monitor. However, the communication must be done within a certain

interval of time, otherwise the data become worthless. This time depends on the sensor. When the communication is accomplished, the sensor sleeps until the next reading.

The monitor waits for both sensors to communicate the data. When the data is received, it takes 1 millisecond to compute the function and then prints the result.

In Figure 1 we show the ATP specification of the system, where the communication function is $data_1!|data_1? = data_1$ and $data_2!|data_2? = data_2$. ■

3 Timed automata

In this section we briefly describe timed automata as they are defined in [8].

We start out with a set X of timers ranging over a time domain D. V denotes the set of timer valuations, i.e., the set $[X \to D]$ of mappings from X to D. For $x \in X$, $x := 0$ is the assignment of the value 0 to x, called reset of x. We denote by $\mathcal{R}$ the set of lists of resets of this type. The set Φ of timing constraints is defined by the following grammar:

$$\phi ::= x \# c \mid x + c \# y + d \mid \neg\phi \mid \phi_1 \wedge \phi_2$$

where $x, y \in X$, $c, d \in \mathbb{N}$ and $\# \in \{<, \leq, >, \geq, =\}$.

An timed automaton is a tuple $G := \langle S, X, s_0, \mathtt{inv}, \to \rangle$where:

S	is a finite set of states,
X	is a set of timers,
s_0	is the initial state,
$\mathtt{inv} : S \to \Phi$	associates a timing constraint to each state,
$\to \subseteq S \times (\Phi \times Act \times \mathcal{R}) \times S$	is the set of transitions.

A timed automaton starts at state s_0 with all its timers initialized to zero. The states of the automaton represent the control states. Moving through an edge (s, ϕ, a, R, s') takes no time. A move sets to zero the values of the timers in R. The values of all the timers increase uniformly with time and, at any instant, the value of a timer is equal to the time elapsed since the last time it was reset. The automaton can perform a move only if the timing constraint ϕ associated to the edge is satisfied by the current values of the timers. The automaton may stay at a given state s but it cannot let time pass beyond the bound imposed by the associated safety timing constraint $\mathtt{inv}(s)$. At any instant, the state of the system can be fully described by specifying the current state of the automaton and the value of all its timers. The formal operational semantics of a timed automaton is given in [8, 7].

4 Graph of timers

A timer system T is defined to be the tuple $\langle X, \mathtt{lab}, \mathtt{type}, \mathtt{upbd}, \mathtt{comm} \rangle$, where X is a finite set of timers, $\mathtt{lab} : X \to Act$ is a partial function that maps each timer to an action name, $\mathtt{type} : X \to \{\mathtt{ac}, \mathtt{to}, \mathtt{wt}\}$ determines the type each timer, $\mathtt{upbd} : X \to \mathbb{N}_\omega$ represents constant upper bounds on timer values (when $\mathtt{lab}(t) = \omega$, there is no finite upper bound)

and $\mathbf{comm} : 2^X \to Act$ is a partial function mapping set of timers to action names (i.e., the communication function).

For instance, the timer system of the sensor process of Example 2.1 is shown in Figure 2: t_4 is associated to the watchdog construct which determines the periodicity of reading, t_2 represents the timeout on the delay of sending data to the monitor, which is modeled by the t_1.

timer	lab	type	upbd
t_0	*sample*	ac	0
t_1	*data_s*	ac	ω
t_2		to	2
t_3	*data_s*	ac	0
t_4		wt	10

Figure 2: Timer system of the sensor process.

A graph of timers G is a tuple $\langle T, X^0, \succ, \triangleright \rangle$ where T is a timer system, $X^0 \subseteq X$ is the set of initial timers and $\succ \subseteq X^2$, $\triangleright \subseteq X^2$.

The graph of timers of a term is obtained by a sintax-driven construction. Figure 3 shows the graph of timers of the sensor process. Notice that, if the data is sent before a time equal to $\mathbf{upbd}(t_2)$, denoted by the expiration of t_1, then t_2 should be stopped but not t_4. When the timeout expires, because a time equal to $\mathbf{upbd}(t_2)$ elapsed, the action *data_s* becomes enable and must be performed immediately. The enabling of the action corresponds to reset t_3. Since $\mathbf{upbd}(t_3)$ is equal to 0, then it expires immediately without letting time pass. This situation is captured by the relation $\triangleright$, $t \triangleright t'$ means that t' is reset when t expires. In the example, $t_2 \triangleright t_3$.

	t_0	t_1	t_2	t_3	t_4
$\succ$			$\{t_1\}$		$\{t_2\}$
$\triangleright$	$\{t_1\}$		$\{t_3\}$		$\{t_0\}$

Figure 3: Graph of timers of the sensor process.

Consider, for instance, the graph of timers of the sensor process (Figure 3). The timer t_0 is the initial timer. When it expires, t_4 is reset and also t_2 and t_1.

Now, consider the scenario where t_4, t_2 and t_1 have been reset simultaneously. When t_1 expires, because its associated action is performed, t_2 must be stopped, but not t_4 since it is a watchdog timer. Hence, in the "following" scenario only t_4 remains. We define the relation $\not\succ \subseteq X^2$:

$$\frac{x \succ x'}{x \not\succ x'} \qquad \frac{x \not\succ x' \wedge x' \succ x''}{x \not\succ x''} \qquad \frac{x \not\succ x' \wedge x' \triangleright x'' \wedge x \neq x'' \wedge \mathbf{type}(x') \neq \mathbf{ac}}{x \not\succ x''}.$$

The timed automaton of a term is $\langle S, X, s_0, \mathtt{inv}, \rightarrow \rangle$ where $S = 2^X$, $s_0 = X^0$ and $\rightarrow$ is constructed starting at the set of initial timers, applying the following rules for any timer t in the current state: (1) if t is an action timer, then all the timeout timers t' such that $t' \gg t$ are removed, (2) if t is a timeout or watchdog timer, then all the timers t' such that $t \gg t'$ are removed, (3) for all t' removed, we add the timers t'' such that $t' \triangleright^* t''$.

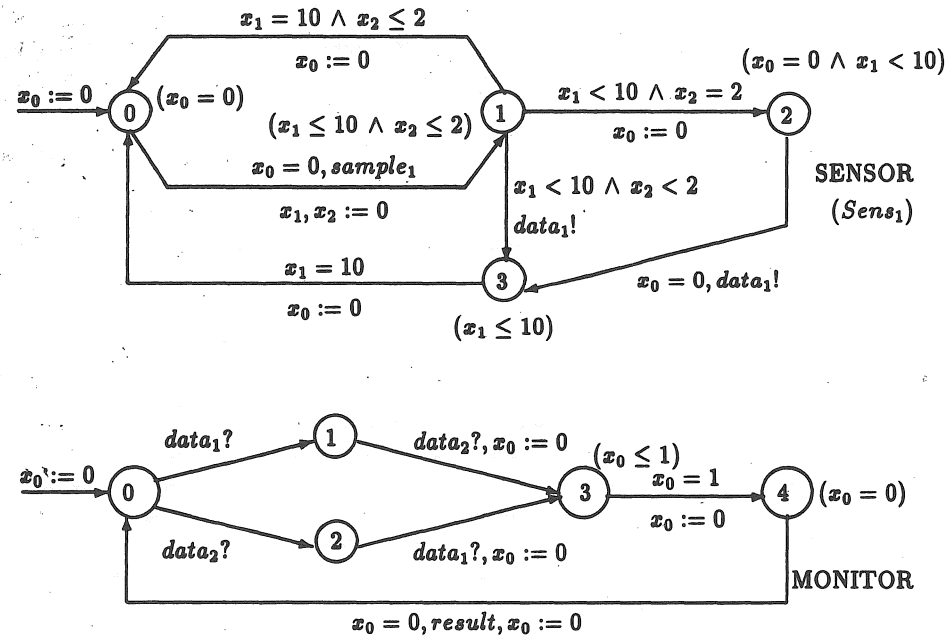


Figure 4: Timed automata for the monitoring system.

5 The tool

We discuss here some implementation details of the existing tool.

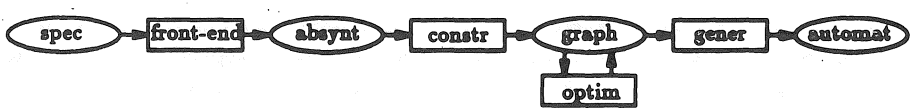


Figure 5: The functional architecture of the compiler.

The functional architecture of the compiler is shown in figure 5. The compilation is done in successive steps:

1. The front-end performs standard syntactic and semantic analysis, expands recursive definitions and constructs an attributed abstract syntax tree, where timers are associated to actions, timeouts and watchdogs.

2. The second phase constructs the *graph of timers*, following a syntax-driven definition.

3. The optimization phase reduces the number of timers by applying standard techniques for the allocation and assignment of registers during code generation (Fig 6).

4. The generation phase generates the corresponding timed automaton by performing some kind of reachability analysis. A state of the automaton is a set of timers. The generation phase mainly consists of three components: the first one computes for a given state of the automaton the outgoing transitions and successor states; the second component is the management of a fast-access extensible hash-table for storing and searching states; the third one consists in the breadth-first construction of the automaton, starting at the initial state.

class	timers
x_0	$\{t_0, t_1, t_3\}$
x_1	$\{t_4\}$
x_2	$\{t_2\}$

Figure 6: Optimization of timers.

Figure 4 shows the timed automata of the monitoring system processes generated by the tool. The compiler handles time parameters of the time constructs symbolically. That is, in the example, the timed automaton of sensor $Sens_2$ is the same as the one of $Sens_1$ shown in the figure, except that timer x_2 is compared to 4 instead of 2.

6 Conclusion

We have briefly presented ATP and timed automata, and illustrated their use with a very simple example. One interesting property of the compilation method is that the time parameters of the time constructs are handled symbolically. Consequently, the size of the timed automaton is completely independent of time constants. The current version of the tool is made up of approximately 11,000 lines of C code. It has not been yet used to compile large specifications. However, the performances achieved for the examples considered appear to be satisfactory.

References

[1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real time systems. In *Proc. 5th Symp. on Logics in Computer Science*, pages 414–425, IEEE Computer Society Press, 1990.

[2] R. Alur and T. Henzinger. Logics and models of real-time: a survey. In *Proc. REX Workshop "Real-Time: Theory in Practice"*, Lecture Notes in Computer Science 600, Springer-Verlag, the Netherlands, June 1991.

[3] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 995–1072, Elsevier Science Publishers (North-Holland), 1990.

[4] C. Gehzzi, D. Mandrioli, S. Morasca, and M. Pezzè. A unified high-level Petri net formalism for time critical systems. *IEEE Transactions on Software Engineering*, March 1991.

[5] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. In *Proc. 7th Symp. on Logics in Computer Science*, IEEE Computer Society Press, 1992.

[6] X. Nicollin and J. Sifakis. *The algebra of timed processes ATP: theory and application.* Technical Report RT-C26, LGI-IMAG, France, December 1990. To appear in *Information and Computation.*

[7] X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *IEEE TSE Special Issue on Real-Time Systems*, September 1992.

[8] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. In *Proc. REX Workshop "Real-Time: Theory in Practice"*, Lecture Notes in Computer Science 600, Springer-Verlag, the Netherlands, June 1991. To appear in *Acta Informatica.*

[9] G.M. Reed and A.W. Roscoe. A timed model for Communicating Sequential Processes. *Theoretical Computer Science*, 58:249–261, 1988.

[10] Wang Yi. Real-time behaviour of asynchronous agents. In J.C.M. Baeten and J.W. Klop, editors, *CONCUR 90: Theories of Concurrency*, Lecture Notes in Computer Science 458, Springer-Verlag, 1990.

Viewpoint Oriented Real-Time Parallel Systems Development

ZHANG Lichen, GALINDO Michel, MARQUIE Daniel and RAYNAUD Yves
IRIT/SIERA, UPS
118 Route de Narbonne
31062 Toulouse,France

ABSTRACT: In this paper, we propose a new approach to real-time parallel systems development which explicitly avoids the use of a simple framework to design complex systems. Instead, this new methodology is proposed according to five views: environment, function, behavior, performance, material and besed on the partition of the system deveopment life cycle into six stages. In each stage, specific techniques are applied.

KEY WORDS: software engineering, viewpoint, real-time, parallel processing.

1. INTRODUCTION

Typical real-time systems involve complex sequences of events, actions, conditions and information flow, often with explicit timing constraints. Failure to meet timing constraints could cause a catastrophe. Distributed systems will likely be the best accepted solution for throughput enhancement of large real-time applications composed of highly concurrent processes, but this does not mean that timing constraints will be met automatically. In fact, the following types of constraints and requirements arise in the development of real-time parallel systems: decomposition of system into tasks, timing constraints, resource constraints, precedence relationships, concurrency constraints, communication requirements, allocation the elementary functions on parallel architecture, even utilization of each processor.

It is fair to say that the problem of finding good methods to aid in the development of such systems has not been satisfactorily solved. Some methods have been proposed for real-time system design [1][2][3][4][5][6], but they are not suited for complex system.

These methods have the gap beween the levels of specification, software architecture and hardware architecture. In addition, they do not provide a solution for guaranting the deadline of real-time tasks. It is in this context that we have been studying real-time parallel system design since 1989 under contract supported by French National Space Study Centre (in French, Centre National d'Etudes Spatiales (CNES)). The project is organized into three phases: state of the art, proposition of an original design methodology, and validation of the solution proposed on an application of CNES.

This paper is structured as follows. The section two depicts our modeling methodology by emphasizing the main aspects and techniques of each stage. The following section overviews the application that is used to validate the proposed methodology.

2. DEVELOPMENT METHODOLOGY

In the second phase, we have proposed a design methodology of parallel real-time systems according to five views: **a view of environment of system, a view of function of system, a view of behavior of system, a view of performance of system, and a view of material of system.** The methodology is based on the decomposition of the system development life cycle into six stages. In each stage, specific techniques are applied.

Stage 1: definition of requirements: The first stage is concerned with determing what is to be built and delivered, how fast il must operate, how good it must be , with what it will interface, how much it will cost and most important, what part the software will perform. Techniques for specifying requirements formally do exist but do not suit for complex real-time system. Instead, requirements will be expressed in **structural nature language [14].**

Stage 2: analysis of the environment of system: The second stage is concerned with: stating the objects concerned by the environment of system, determing the operations, events, informations concerned by the objects, describing the behaviour of each object.

The objects of environment have the following characteristics: internal strength cohesion and weak coupling between the objects. Therefore, **the HOOD [5] /ESTEREL [6] integration or HOOD/SDL [7] integration** are adapted for analyzing the environment. HOOD is used for the identification of objects of the environment, and ESTEREL or SDL are used for the behavioural description of the elementary objects. ESTEREL is reserved for critical applications, whereas SDL is adapted for less critical applications.

Stage 3: refinement of the system: The refinement strategy is globally top-down and consists in the execution of a set of basic steps where a given function is broken down into sub-functions. Each such sub-function is in turn defined and broken into other lower level components in following refinement step. A top-down decomposition of a system stops up to elementary functions. In this stage, we make **the extension on the method SADT [13]** to permit expression of three types of relations between functions: synchronization by event, represented by the line ——————, transfer of information by variable of state, represented by ----------, transfer of information by channels, represented by ———.

Stage 4: analysis of the behavior of system: We describe the behavior of system with its environment. The control activites can be specified by the behavioral view. These must be present on any level of the activity hierarchy, controlling that particular level. These controllers are responsible for specifying when, how and why things happen as the system reacts over time. We make use of the **Statecharts [4]/PRTIL [9] integration** to analyze the behavior of system. Statecharts, extensions to Finite State Machines (FSM), were recently proposed by Harel. Statecharts make it even easier to model complex system behavior without ambiguity. The extensions provide a notation and set of conventions that facilitate the hierarchical decomposition of FSMs and a mechanism for communication between concurrent FSMs, the transitions depend on global conditions and on being in particular states. PRTIL is used for specifying the condition that guards the transition.

Stage 5: evaluation of performances: During this stage, the analysis and simulation models of the system may be developed, based on a user defined multi-processors or a hypothetical multi-processors. These models help determine performance bottlenecks in the system and assist in selecting the hardware configuration. Rough estimates of processing times and communication times for all the elementary functions are developed, generally based upon lines-of-code estimates. These estimates are tallied to verify whether system fits within harware and processing timeliness constraints.

In order either to simplify the model so that an analytic method is made applicable, or to reduce the number of event to be simulated, **aggregation methods [10]** are particularly well suited to parallel architecture modeling, since these architecture are very often composed of similar, even idential, components that constitute choice aggregates.

Stage 6: implementation: Firstly, some constraints related to design solution are introduced for refining the functional structures. Secondly, the repartition between hardware and software is determined according to the results of evaluation of performances, the hardware must be specified. After the repartition between hardware and software, the elementary functions are allocated among parallel processors to achieve following goals: meeting all timeliness constraints, meeting precedence constraints between functions, minimizing communication cost between processors, balancing utilization of each processor. The above allocation meets certain optimization criteria without specifing the order of execution of functions on a processor, we can use such an allocation pattern, in conjunction with heuristic rules for ordering task allocated to each processor and the communication and synchronization between functions on a processor can be realized by hardware or software. The realization of the communication and synchronization between processors depends on the nature of relation between the functions.

The heuristic method is to provide fast and effective algorithms for a suboptimum solution of allocation. It is useful in application where an optimun solution is not obtainable within a critical time limit and is also applicable to larger dimensional problems. We make use of a function allocation model based on a **zero-one programming technique, the branch-and-bound method [11]**. The design of a model for allocation involves the following steps: 1) formulate the cost function to measure the interprocssor communication cost and processing cost by using the results of evaluation of performance, 2) formulate a set of constraints to meet the diverse requirements by using the results of evaluation of performance, 3) derive an iterative algorithm to obtain a minimum total cost solution.

The relation "variable of state" between functions is realized by memory. The mutual exclusion is commonly employed when using shared variables. The relations " synchronization " and " transfer of information by channels" are realized by interrupt or boolean variables or semaphores or physical channels.

3. EXAMPLE OF APPLICATION

An application has been proposed by CNES for validation of the methodology proposed in the second phase, and the prototype is constructed by use of Transputers and OCCAM [8].

(1) SYSTEM REQUIREMENTS

The simulator must be able to provide the transfer frame that complies to the CCSDS Recommendations and is identical with the frame which is produced by an onboard generator. The primary requirements for this simulator are as follows: 1) to simulate application data and to create source packets, 2) to multiplex these different packets on a virtual channel, 3) to generate the transfer frame, 4) to add CRC or R-S code, and insert the Synchronisation marker and serialize message for generating Physical Channel Access Protocol Data Unit. The Figure 1 specifies the primary functions of system by method SADT (13).

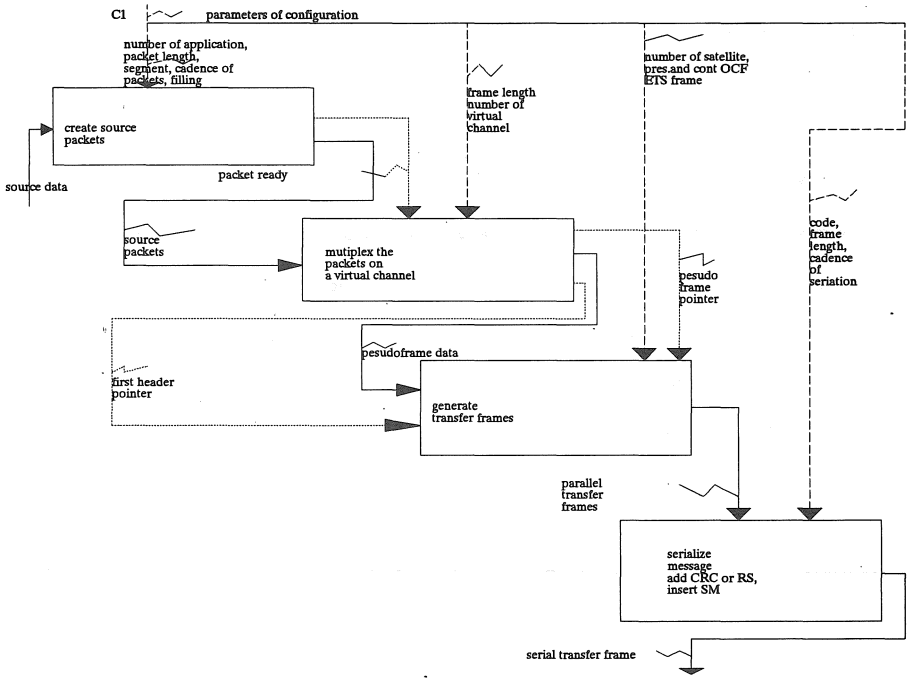


Fig. 1 primary requirement of the simulator

(2) HARDWARE AND EXECUTIVE SOFTWARE

In our simulator, the system consists of five transputers made up from a Volvox 1/A board [5], which fits inside an IBM PC AT, containing one T425 processor and a Volvox 4/A containing four T425 processors. The executive software is written in OCCAM language. In the simulator, the fourth function is realized by a serial generator. This hardware communication with Transputer network is by an Inmos C012 link Adaptor.

(3) COMMUNCATION PERFORMANCE ANALYSIS

One important characteristic of the application is the great communication amount between the Transputers. This causes message transfer delay. We evaluate the influence of various parameters on message delay. The following factors are of great importance to message delay (Fig. 2, Fig. 3, Fig. 4): the number of processes on a processor, message size, probability of transfer message of processes.

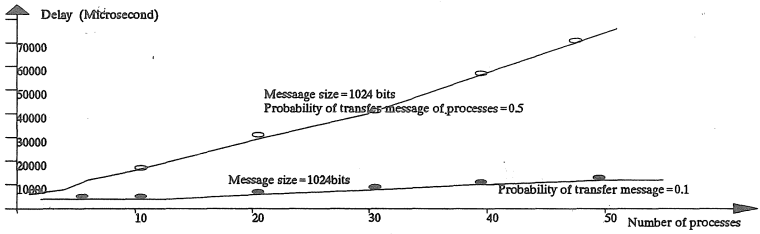


Fig. 2 Delay in function of number of processes

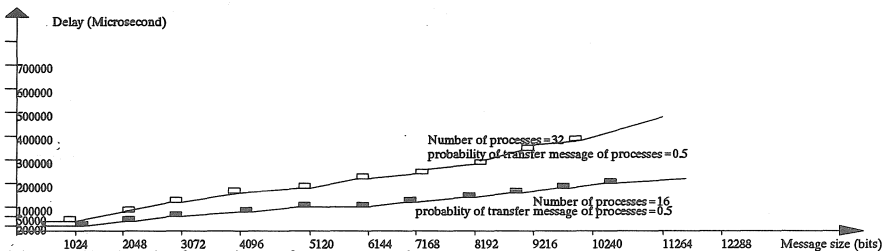


Fig.3 Delay in function of message size

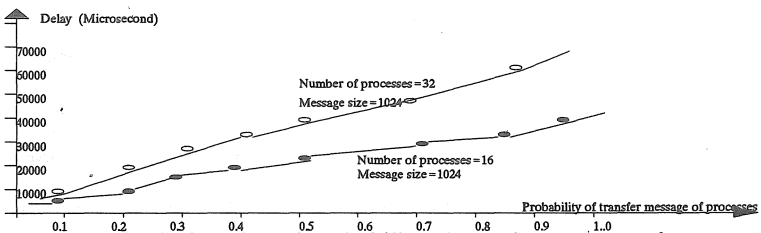


Fig. 4 Delay in function of probability of transfer message of processes

(4) ALLOCATION

Any allocation must first satisfy the communication requirement between the Transputer because the communication cost is greater than the processing cost in our application. Since the processing cost and communication cost is scenario dependent, the allocation can be optimized only for a given scenario.

4. CONCLUSION

In conclusion, we may say the our methodology is adatpted for real-time parallel system design because we do not attempt to use a simple framework to design complex system. The application has shown that the proposed methodology is complete, the results of evaluation of performances are useful for allocation of processes on multiprocessor and provide the solution for garanting the deadline of a real-time task.

REFERENCES

1. M.Jackson, System development, C.A.R HOARE series, Prentice-Hall 1983.

2. Joint IECCA and MUF Committe on MASCOT (JIMCOM), The Official Handbook of MASCOT. Her Majesty's Stationery Office (June) 1987).

3. P.T. Ward and S.J.Mellor, Structured development for real-time systems, Yourdon Computing series-YOURDON PRESS-Prentice-Hall 1985.

4. D.Harel, Statecharts, a visual formalism for complex systems, Science of computer programming North Holland, Vol.8, 1987, P.231-274.

5. HOOD Working Group, HOOD Eeference Manual, draft B issue 3.0 (June 1989).

6. G.Berry and G.Gonthier: "The synchronous language ESTEREL: Design, Semantics and Impementation" INRIA report 842, 1988.

7. CCITT: "Fonctional specification and description language (SDL)", Recommandations Z.101-Z.104, Vol. VI, Fasc. VI.7, Genova 1981.

8. E.Hirsh, Les Transputers: Application à la programmation concurrente, Editions Eyrolles 1990.

9. K.T. Narayana and A.Aaby, Specification of real-time systems in real-time temporal interval logic, Proceeding of real_time systems symposium, IEEE Computer Society Press, 1988.

10. G.Pujolle and S.Fdida, Modèles de systèmes et de réseaux,Editions Eyrolles, Paris, 1989.

11. P.Richard, Y.Edward and M.Tsuchiya, A task allocation model for distributed computing systems, IEEE Transactions on Computers 31(1): 41-47,1982.

12. CCSDS 102.0-B-2, "Recommendation for Space Data System standards. Packet Telemetry.", CCSDS Recommendation, Blue Book, Issue-2, January 1987 or later issue.

13. I.G.L. Technology, SADT, Un langage pour communiquer, Editions EYROLLES, 1989.

14. D.J.Hatley and I.A.Pirbhai, Strategies for real time systems, Rourdon computing series-Yourdon Press Prentice Hall 1985.

VSKEP-EDITor: A Visual Tool for Editing Reusable Skeletal Plans

Wu Zhaohui and Bernardi Ansgar *
German Research Center of Artificial Intelligence,
Postfach 2080, Kaiserslautern 6750,
Germany
email: {wu,bernardi}@dfki.uni-kl.de

Abstract

In order to reduce the acquisition task of expert knowledge about process planning in manufacturing, which are represented as skeletal plans in the PIM system, we propose to implement a visual editing tool called VSKEP-EDITor. This tool not only can visualize the structure of skeletal plan and automatically check the syntax of skeletal plan when editing it, but also provides a taxonomic hierarchy of existing skeletal plans for reusing them to create new ones. These reusable skeletal plans can be adapted as the edited candidates to greatly reduce the task of knowledge acquisition.

1 Introduction

The PIM [1][2] system (Planning in Manufacturing) is a knowledge-based Computer Aided Process Planning (CAPP) prototype, in which a group of representation languages and formalisms are provided to fill the gap between

*The work is as part of the ARC-TEC project, supported by BMFT under grant ITW 8902 C4.

CAD and CAM. In these formalisms, SKEP_REP is a language for representing expert knowledge about process planning in manufacturing. Based on these expert knowledge in the form of SKEP_REP, the proposed PIM system can generate a production plan by performing a sequence of abstraction, selection and refinement. Therefore, Crafting these knowledge into PIM becomes more important.

After having analyzed the process planning in manufacturing and its presentation language SKEP_REP[3], we found the following features:

- There exists great redundency among these skeletal plans and some new skeletal plans are acquired only through limit modifications on these existing skeletal plans.
- These skeletal plans formalized by the SKEP_REP language are closely related to their associated manufacturing feature structure[4], the visualization of these feature structure makes expert well understand about these knowledge.
- The SKEP_REP formalism is a language with a restrict terminology and a simple syntax template.

Therefore, in order to effectively reduce the acquisition task of expert knowledge in the form of SKEP_REP, we propose to implement the system called VSKEP-EDITor (Visual SKEletal Plan EDITor), and to integrate the proposed tool VSKEP-EDITor into the Common Lisp based PIM system, we adopt the following technical strategies or principles:

- to implement it by using the mature and powerful user interface programming facility CLIM. The CLIM (Common Lisp User Interface Management System) is a Advanced User Interface development system in Symbolics[5][6], which provides a group of facilities about program interface controlling, user input and formatting output to help users to approach the user interface of his/her application programs. These facilities will greatly reduce the developing task of VSKEP-EDITor.
- to analyze the skeletal plan representation formalism SKEP_REP and to find out the template of representation and the visual structure of its feature structure. The simple version of SKEP_REP is described in the section 2.

- to investigate the organization structure of those existing skeletal plans for effectively reusing them to create a new skeletal plan. A taxonomic hierarchy of skeletal plans is shown in the section 3.

We are to introduce the architecture and corresponding mechanisms of this tool in section 4 and to illustrate its characterisics in section 5.

2 SKEP_REP: A Skeletal Plans Formalism

SKEP_REP a formalism for representing skeletal plan, which is used to describe expert knowledge about process planning on special part or feature of workpieces. A skeletal plan formalized in SKEP_REP consists of the following components:

- Op-Mode — to describe the operational mode of a skeletal plan.
- Special-Tree — to describe the feature structure of a skeletal plan.
- Constraints — to describe the local constraints of a skeletal plan.
- Actions — to describe the planning actions of a skeletal plan after having satisfied the feature structure and constraints. The actions part of the skeletal plan usually is a set of restrict lisp functions such as chucking, roughing, merging plans actions and so on.

3 A Taxonomic Hierarchy of Skeletal Plans

A taxonomic hierarchy of skeletal plans is built by effectively classifying. For one side, the classification of these skeletal plans can help expert to hierarchical browse all the skeletal plans which had been finished, and also assist him/her to fix the appropriate reused candidates of these skeletal plans in the hierarchy. On the other hand, we enable to add new skeletal plans into the hierarchy and to make these skeletal plans maintainable.

The taxonomic hierarchy of these skeletal plans are built from the partial order ($\succeq$) over these skeletal plans. This partial order is introduced through the partial relation ($\succeq$) over the feature structure contained in skeletal plan and the inclusive relation among the predicates of the local constraint in skeletal plan. We define the skeletal plan order as follows:

- Given two skeletal plan P-a and P-b, P-a contains of the feature structure P-a-FS and the constraint P-a-C, P-b contains of the feature structure P-b-FS and the local constraint P-b-C. the P-a $\succeq$ P-b iff P-a-FS $\succeq$ P-b-FS and P-a-C $\mapsto$ P-b-C.

This partial ordering relation satisfy the following axioms:

- Reflexive: $U \preceq U$
- Transitive: If $U \preceq V$ and $V \preceq W$, then $U \preceq W$.
- Antisymmetric: If $U \preceq V$ and $V \preceq U$, then $V = U$.
- Top: $\top \succeq U$
- Bottom: $U \succeq \bot$

An illustrated taxonomic hierarchy of skeletal plans is shown in figure 2, where the *"TOP-SKep"* is a generalization of the *"SKEP-01"*.

4 Architecture

The architecture of the proposed tool VSKEP-EDITor is illustrated in the figure 1. The basic components of this architecture are[7][8]:

- a controller – controls to.read/interpret/execute the user input command.
- a command table – a extensive command table which provides the functions of VSKEP-EDITor, and can be improved in the future version.
- three independent buffers for skeletal plans, the template buffer, the editing buffer and the reused buffer.
- a classifier — to classify these given skeletal plans into a hierarchy.
- a template-driven acceptor – to accept information from the template of skeletal plan and also to fill these items in the template.
- a lister – to input the user's command.

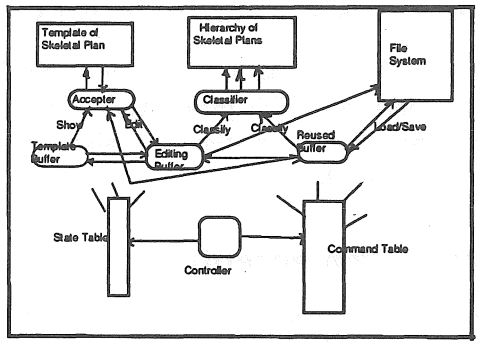


Figure 1: The Architecture of VSKEP-EDITor

4.1 Independent Buffers for Skeletal Plans

As the proposed VSKEP-EDITor supports various operations on skeletal plans stated in different types medium, these require us to implement various pools storing skeletal plans to promote the different requirements of command operations.

The separation of these individual buffers is to simply the complexity of implement and to enrich the functions of VSKEP-EDITor. The follows listed are three kinds buffers for skeletal plan:

- Editing Buffer — to store these skeletal plans which have been edited. This buffer introduced is to support the edit of more than one skeletal plans and each of them can flexibly be added into the hierarchy of skeletal plans.

- Template Buffer — to store these skeletal plans which are used to be displayed or being edited. This buffer is used as a cache for the template of skeletal plan.

- Reusing Buffer — to store these skeletal plans which are loaded from the file system or classified from the other buffers, are illustrated in the hierarchy window and each of them can be selected as the reused candidate for editing a new skeletal plan.

4.2 A Classifier : Forms the Hierarchy of skeletal plan

The classifier illustrated in above architecture, is to classify skeletal plans in these buffers into the hierarchy. This means that it not only can classify the skeletal plans loaded from the file system, but also can classify these edited skeletal plans into the hierarchy.

4.3 Template-driven acceptor

In VSKEP-EDITor, we implement a template to describe the syntax structure of SKEP_REP. The template consists of three subtemplates, the feature structure template, the constraint template and the action template, which are extracted from the syntax of skeletal plan.

The acceptor can access the whole part of proposed template of skeletal plan and enables to display certain plan on this template and to show the text description of feature structure in the feature template in the visual way. The acceptor keeps checking on the syntax of input from three individual templates.

4.4 State-Variable Controller

The controller in the above mentioned architecture, is a simple command loop of read/interpret/execute, in which the controller is to maintain the user's command table and to control the execution of these commands in the command table.

4.5 User Interface

The user interface is a important component of VSKEP-EDITor. The configuration of user interface illustrated in figure 2, consists of the following five parts: (1) A Hierarchy Display Window for Skeletal Plan. (2) Two groups Command Menus. (3) Feature Structure Display Window. (4) Skeletal Plan Template Window, including the following three panes: Feature Structure Template, Constraint Template and Actions Template. (5) A Listener Window.

During using the VSKEP-EDITor tool, we can combine the command menu selection and character accelerator to input a user's command appear-

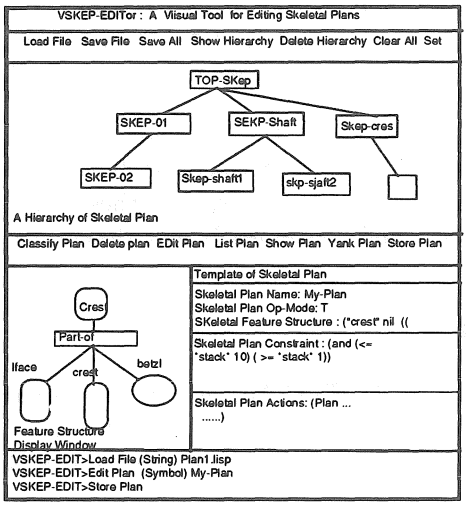


Figure 2: User Interface of VSKEP-EDITor

ing in the listener window; and type-in or choose the output of representation by mouse to fill the arguments of the command such as the command " Edit Plan 'My-Plan ". The skeletal plan is generated by editing or modifying them within the template window.

5 Conclusion

VSKEP-EDITor had been implemented in Commonlisp (totally 60 KB code) and as a tool in the PIM system and has the following characteristics:

- Hierarchical Skeletal Plans Reusable Edit – VSKEP-EDITor can copy and yank the whole or part of these existing skeletal plans and form new one through limit modifications.
- Feature Structure Visualizable Edit – VSKEP-EDITor is to visualize the feature structure of a skeletal plan automatically when editing it.

- Template-Driven and Syntax-Sensitive Edit — VSKEP-EDITor provides a skeletal plan template driver to guide the operations of edit and a syntax automatic checker to cover the user's input.
- Friendlyness of User Interface — VSKEP-EDITor is to implement a command menu, command accelerator, button list, multiple windows, graphic formatting and terminological thesaurus to help expert interactively, friendly and conveniently edit these skeletal plans.

References

[1] Bernardi A, etc:ARC-TEC:Acquisition,Representation and Compilation of Technical Knowledge, In Expert Systems and Their Applications:Tools, techniques and Methods,1991,pp133-145.

[2] Legleitner, R., Bernardi, A,etc:*PIM: Skeletal Plan based CAPP,* Proceeding of Int' Conference on Manufacturing Automation, 1992.

[3] Andres Becker:Analyse der Planungsverfahren der KI in Hinblick auf ihre Eignung fur die Arbeitsplanung, D-91-17, DFKI GmbH, 1991.

[4] Klauck C., Bernardi A,etc:FEAT-REP: Representing Features in CAD/CAM. In IV Int' Symposium on AI:Applications in Informatics,1991.

[5] Symbolics:Programming the User Interface, Book 10, 1990.

[6] Symbolics:User Interface Dictionary, Book 11, 1990.

[7] Zhaohui Wu and Bernardi Ansgar:User Guide of VSKEP-EDITor Tool, DFKI Document, (Forthcoming), 1992.

[8] Zhaohui Wu and Bernardi Ansgar:VSKEP-EDITor: A Visual Tool for Editing Skeletal Plans, DFKI Document, (Forthcoming), 1992.