A.O
17
91
II

CEDINFOR

PANEL'91

XVII Conferencia Latinoamericana de Informática

CNIASE'91

IV Conferencia Venezolana de Inteligencia Artificial y Sistemas Expertos

Volumen II

8 al 12 de Julio de 1991
Caracas
Venezuela

FUNDACION CEDINFOR
ORIGEN
DOCUMENTO SAVIO
NRO. INV. 1243
FECHA 21 OCT 1991

Copias adicionales : Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información, Apartado 89000, Caracas 1080-A, Venezuela
FAX: 58-2-9621695
Tel.: 58-2-9633022 Ext 3320, 9073320
Correo electrónico: emsca!usb!panel@sun.com

XVII Conferencia Latinoamericana de Informática
PANEL'91

Julio 8-12, 1991

organizada por:

Universidad Simón Bolívar

Presidente de la Conferencia

Jorge Baralt (*USB*).

Presidente y Comité de Programa

Julián Aráoz (*USB*) (**Presidente**), Alvaro Galvis (*U. de los Andes, Colombia*), Héctor Geffner, (*IBM Watson Research Center, USA*), Clovis Gonzaga (*UFRJ, Brasil*), Alonso Márquez (*USB*), Consuelo Maulino (*UCV*), Alberto Mendelzon (*U. de Toronto, Canadá*), Mauricio Milchberg (*Bull, France*), Eduardo Pelegrí (*Sun Microsystems, USA*), Roger Soler (*USB*), Patricio Poblete (*U. de Chile*), Jorge Vidart (*ESLAI, Argentina*), Alejandro Teruel (*USB*).

Comité Organizador

Edgardo Broner (**Presidente**), Emely Arráiz, Oscar Meza, Cristina Zoltan de Torres.

Enlace Universitario

Néstor Carrasquero, Francis Losavio, Alfredo Matteo, Rebeca Landau (*UNIMET*), Marta Sananes (*ULA*), María Bravo (*UDO*), Francisco Montes de Oca (*UCLA*), Daniel Bermúdez (*UNEG*), Héctor Arciniegas (*LUZ*), Marta Rukoz (*UCV*), Martha Arteaga, Sim Soon-Kiong (*IUT-RC*), Wilfred Durán (*UC*), Pablo Ríos (*UPEL*).

Comité Directivo del CLEI

José A. Pino Urtubia, *Presidente*
Jorge Vidart, *Past-President*
Marcos Borges, *Secretario Ejecutivo* (Brasil)
Ida Holz Bard, *Tesorero* (Uruguay)
Aldo Migliaro Osorio, *Miembro Honorario*
Jorge Santos (Argentina)
Felipe Guzmán Brockmann (Bolivia)
Consuelo Franky (Colombia)
Ignacio Casas (Chile)
Diego Andrade S. (Ecuador)
Adrián Araya (Costa Rica)
Victoria Bajar (México)
Gerónimo Bellassai (Paraguay)
Edgardo Broner (Venezuela)

Se agradece la colaboración en las revisiones de los trabajos enviados a la

XVII CONFERENCIA LATINOAMERICANA DE INFORMATICA PANEL'91

Galvis, Alvaro
Teruel, Alejandro
Figueira, Carlos
Maulino, Consuelo
Di Prisco, Carlos
Broner, Edgardo
Pérez, Elena
Chiariotti, Enzo
Rodríguez , Eva
Geffner, Hector
Baralt, Jorge
Ramírez, Jorge
González, Luis
Mendelzon, Alberto
Lentini, Marianela
Vidal, María
Meza, Oscar
Planchart, Enrique
Poblete, Patricio
Ravelo, Jesús
Soon-Kiong, Sim
García Palomares, Ubaldo
Fouchet, Claude
Rueda, Francisco
Toro, Victor
Jimenez, Claudia
Cardosa, Rodrigo
Nava, Alberto
París, Bernardo
Gonzaga, Clovis
Zoltan, Cristina
Arráiz, Emely
Fimbel, Eric
Montagne, Eurípides
Pelegrí, Eduardo
Losavio, Francis
Aráoz, Julián
Molero, Jorge
Vidart, Jorge
André, Michele
Meier, Andreas
Rukoz, Marta
Callaos, Nagib
Ordaz, Oscar
O'Callaghan, Patrick
Pino, José
Soler, Roger
Soriano, Amelia
Teoktisto, Víctor
Márquez, Alonso
Lleras, Ernesto
Hernandez, Tiberio
Cortés, Gloria

Colaboraciones

IFIP, CREI, Embajada de Francia, Embajada de España, Decanato de Investigaciones (USB), Decanato de Postgrado (USB), Petróleos de Venezuela y sus Empresas Filiales (patrocinante mayor), CEPET, CANTV, SIDOR (patrocinante), Oficina Regional de Cooperación del Ministerio de Relaciones Exteriores de Francia.

Colaboraciones recibidas hasta 13 de Mayo de 1991.

Mensaje del Presidente del Comité de Programa

Estas actas contienen los 77 trabajos seleccionados por el comité de Programa, de un total de 168 enviados a la **XVII Conferencia Latinoamericana de Informática PANEL'91**. Asímismo se incluyen resúmenes de algunos trabajos que el comité consideró de interés.

La décimo séptima edición de la Conferencia del **CLEI** es una prueba más del esfuerzo continuado de la región para intercambiar experiencias que permitan elevar los niveles de la Informática en Latinoamérica. Se recibieron trabajos procedentes de las dos terceras partes de Latinoamerica y de países industrializados, lo que evidencia tanto el reconocimiento que ha alcanzado la conferencia, como los logros del **CLEI** en cuanto a la integración en la región.

Esta conferencia del **CLEI** comenzó en Valparaíso, Chile, en 1974. En 1980, Caracas fue la primera sede fuera de Chile. Nuevamente se realiza en Caracas, y el año próximo cruzará el Atlántico para celebrarse en Las Palmas, Islas Canarias, con motivo del V Centenario del encuentro de dos mundos.

Superadas exitosamente las dificultades de correo y comunicación, mal endémico en la región, que dificultó la tarea de los revisores, aprovecho la oportunidad para agradecerle, a cada uno de ellos su dedicación y puntualidad que hicieron posible la edición de estas actas.

Julián Aráoz
Presidente
Comité de Programa

Tabla de Contenido

Bases de Datos

Modelo para la Evaluación de Rendimiento de Sistemas Manejadores de Base de Datos Distribuidos 1151
Judith Barrios Albornoz

Formas Normales Modelo Entidad Relación 1283
Alonso Márquez, Ma. Esther Vidal

S.O.O.M. Un Modelo Semántico para Bases de Datos 1251
Augusto Giancola, Francisco Marcano, Leonid Tineo, María Esther Vidal, Enzo Chiariotti

Uma Experiência na Implementação de Operadores da Álgebra Relacional no Computador Paralelo NCP I 681
Marta L. Queirós Mattoso, Claudio Luis De Amorim

Um Sistema de Interface Cooperativo para Modelagem em Banco de Dados 1103
Luís Eduardo Saraiva Câmara, Décio Fonseca

BENCHREL - Benchmark para a Avaliação de Desempenho de SGBD's Relacionais 1087
Maria Antonieta Rossatto

Computación, Educación y Sociedad

Herramientas y Metodología para el Desarrollo de Aplicaciones Académicas 111
Adelaide Bianchini T., Edna Ruckhaus de Millán

Personal University: Building an Academic Environment 183
Giovanni Degli Antoni, Riccardo Ferrari

Adquirindo Vantagem Competitiva a través de Recursos de Informação 1009
Luiz Henrique Boff, Ricardo Teiltelroit, Êrica Rosalba Mallmann Duarte

Ingeniería de la Programación

Características Determinantes das Tecnologías de Desenvolvimento de Software 851
Vera Maria Benjamim Werneck, Teresa Cristina de Aguiar, Ana Regina C. Da Rocha

Análise da Metodologia JAD – Joint Application Design, de Modelagem de Sistemas: Do Paradigma à Aplicação 1029
Ricardo Teitelroit, Luiz Henrique Boff

Un Estudio de Técnicas de Estructuración de Software: Aplicación en el Ambiente ADELE 281
N. Belkhatir, R. Casallas, W.L. Melo

Un Ambiente para la Enseñanza de la Programación en Lenguaje Ensamblable 41
Enzo Chiariotti G., David A. Koufaty A.

Forum: Una Herramienta para la Gestión de Información sobre Problemas, Terminologías y Referencias en Teoría de Grafos 567
Carlos Gálvez, Francis Losavio, Oscar Meza, Oscar Ordaz

Ambiente de Especificación Denotacional 1173
Pedro Borges, Roger Soler

A Descrição de uma Interface Gráfica e Interativa para Matemática Computacional 949
Álfio Ricardo Martini, Tiaraju Asmuz Diverio

Another Look at Program Visualization and Algorithm Animation 409
Ricardo A. Baeza-Yates

Organización y Documentación de Programas Grandes en 'C' (ANSI) 525
Víctor Manuel Toro C.

Inteligencia Artificial: Adquisición y Representación del Conocimiento

Criterios de Selección para las formas de Representación del Conocimiento 1223
Elena Pérez Miñana, Alejandro Teruel

Multis: A Knowledge Acquisition System Based on Problem Solving Primitives 21
Yuri Adrian Tijerino, Tadahiro Kitahashi, Riichiro Mizoguchi

Sistemas Evolutivos: Nuevo Paradigma de la Informática 1131
Fernando Galindo Soria

Connectionist Approach for Representation of Uncertainty in Knowledge Bases 623
José Ramírez G., Irene Torres Hecker

Una Metodología de Adquisición de Conocimiento para Sistemas Expertos de Diagnóstico Médico 509
Enrique Hinostroza, Pedro Hepp, Jorge A. González, Julián Mardones

Knowledge Adquisition Combining Analytical and Empirical Techniques 235
Ramón Sangüesa, Mario Martín, Ulises Cortés

O uso de Semântica na Construção e Expansão de Árvores de Decisão Indutivas 193
Giuseppe Mongiovi, Walfredo Da Costa Cirne Filho

Inteligencia Artificial: Herramientas y Experiencia

ASTUTO – Uma Arquitetura para o Desenvolvimento de Sistemas Blackboard em Prolog 607
Gilberto Shigueo Nakamiti

3IEN: Un "Shell" de Sistemas Expertos Múltiples que Recibe un Texto de Entrada 493
Manuel González Hernández, Francisco Galán Jiménez, Jesús González Cuevas

Programming Environment for Expert System Shell 249
Wided Lejouad

Hipermedio para Apoyar Explicaciones de Sistemas Expertos 85
Andrés Lepe A., Pedro Hepp K.

Un Sistema Experto para el Diseño del Modelo Conceptual de una Base de Datos 547
Jorge L. Del Mar Arzola

Aplicación de la Tecnología de Sistemas Expertos al Diagnóstico de Daños en Estructuras Afectadas por Incendios 905
Michele Tatoli, Tomás R. Maneiro, Miguel Cerrolaza

Diseño de Circuitos Electrónicos Basado en Reglas 223
Javier Cañas R., Ernesto Lux G.

DIANA: Sistema Experto para el Diagnóstico de Fallas en Miniplantas Compresoras de Gas 205
Rafael Ramírez, Daniel Martínez, Luis Lugo, Mike Backhurst

EXCAYA: Ayuda Experta en Caracterización de Yacimientos 783
Nicelia Hidalgo, Yineira Montilla, Elly G. De Morros, Roberto González T.

Inteligencia Artificial e o Controle de Processos 477
Mauricio G. Solar F., Sueli B. T. Mendes

Inteligencia Artificial: Inferencia e Incertidumbre

Default Reasoning as a Decision Process on Inductive Inferences 749
Fernando Tohmé, Claudio Delrieux

Choosing among Uncertainty Management Techniques: A Case Study 729
Alessandro Saffiotti, Elisabeth Umkehrer

Un Experimento en Aprendizaje Automático y Evaluación Parcial 437
David A. Fuller, Sacha Bocic

Acumulación y Desacumulación de Evidencia en Sistemas de Mantenimiento de la Verdad 1119
Ramón García Martínez, Antonio Marsiglio

Un Modelo de Desacumulación de Evidencia para Sistemas Expertos 403
Ramón García Martínez, Antonio Marsiglio

Search in Horn's Knowledge Bases and the Simplex Algorithm 1273
Julián Aráoz

Inteligencia Artificial: Redes Neurales

Redes Neurales Capacitivas 641
Eric Fimbel

A Neural Network Based Approach to Sensor Fault Detection 131
Abhay Bulsari, Alexander Medvedev, Henrik Saxén

Un Circuito de Redes Neurales para la Interpretación de Patrones Ambiguos Mediante la Extracción de Autovectores 151
Leopoldo Mauro, Andreas Meier

Paralelismo, Sistemas Distribuidos y Optimización

Algoritmos Paralelos para Búsqueda de Patrones: Preproceso del Patrón 57
David A. Koufaty A., Alejandro Teruel L.

Paralelización de un Problema de Aplicación sobre una Plataforma de Transputadores 1
Hector Klie, Alejandro Teruel

SIMplex PARalelo (Simpar): Fase I 991
Alfonso Reinoza, Marianela Lentini, Alejandro Teruel, Lidia Da Silva, Alfredo Guillen, Tito Lozada, Oscar Naím

Multifolding: An Efficient Mapping of Parallel Algorithms onto Array Processors 389
Walter Cunto, Jorge H. Gonçalves R.

Un Manejador de Comunicaciones Minimal para Programas Síncronos sobre Redes de Transputers 1239
Carlos Figueira

Implementación de un Algoritmo de Puntos Interiores para Programación Lineal 99
Marielba Rojas, Marianela Lentini

Un Algoritmo para el Problema de Asignación Generalizado con Múltiples Recursos 813
Alejandro Crema

Creación Dinámica de Procesos en Linda 369
Leonardo Balbiani, Mónica Bobrowski

Derivación de Secuencias de Test para Protocolos de Comunicaciones 927
Leonardo Marty, Fernando Brum

Um Sistema para o Trabalho Cooperativo Baseado em Rede Local 763
Luiz Fernando Bier Melgarejo, André Luiz Costa Ballista, Márcio Quintaes Marchini

La Evaluación de Desempeño en una Metodología de Desarrollo de Sistemas de Información Distribuidos 295
José Aguilar, Alberto Valderruten

Distributed Implementation of a Linda Kernel 313
Rogelio Alvez, Sergio Yovine

Diseño y Evaluación de un Servidor de Recursos para una Red Local 459
Francisco Aurtenechea O., Ignacio Casas R., Roberto Cumsille U., Marco Kútulas P.

Procesamientos De Imagenes

Multiple Fonts from a Single Font Optical Character Recognizer 167
Riccardo Ferrari, Malvina Duarte, Carlo Viscomi

ADI: Ambiente de Desarrollo de Aplicaciones para Procesamiento de Imágenes Digitales 891
Idalia C. Solis Castro, Ma. Luisa Herrera Lopez, Edgar Gerardo Mondrágon Martínez

Técnicas Avanzadas para la Generación de Imagenes 793
Carlos Enrique Alvarez Guerrero

Robótica

Un Lenguaje Implícito Basado en Rasgos para la Programación de Robots 419
Jorge E. Morales V.

MSIM: Un Ambiente de Programación de Robots con Capacidad de Planificación de Tareas 835
Jorge A. Villalobos S.

ROBCAD, C.A.D. para Robótica 1069
Osvaldo Alberto Driollet

Sistemas Formales

Capacidades de Especificación, Análisis y Verificación de Propiedades de Supervivencia y Vitalidad de Sistemas Distribuidos Sinergésicos del Escenario SINERGIA_w 587
Charles R. Páez, Leandro R. León

Systematic Program Development from Abstract Data Type Decomposition. A Case Study: The Topological Sort on Oriented Graphs 1203
Francis Losavio, Alfredo Matteo, Françoise Schlienger

Transformaciones de Programas. Derivación de Versiones Aplicativas Eficientes 871
Liliana Favre

Sobre Especificaciones y Modelos 1189
C. Zoltan de Torres

First Order Liveness Analysis Via Type Inference 1305
Juan Carlos Guzmán, Paul Hudak

On Formally Refining Timed Petri Nets 659
Sergio Bandinelli, Esteban Feuerstein

Communicating Petri Nets 267
G. Degli Antoni

Sistemas Orientados Por Objetos

A Note on The Definition of Visibility for Protected Class Members in C++ 77
Alberto Nava B., Enzo Chiariotti

Modelagem de Sistemas de Manufatura: Uma Abordagem Orientada a Objetos 971
Luis Fernando Friedrich

Projeto Conceitual e Lógico de Sistemas de Gerência de Banco de Dados Orientado a Objetos 713
Cláudio N. F. Trotta, Cláudia M. L. Werner

Uma Representação Gráfica para o Modelo de Objetos do GEOTABA 697
Luiz Carlos Montez Monte

A Formal Specification for a Database Model with Object Oriented Features 353
Regina Motz, Décio Fonseca

Programming with Structures, Functions, and Objects 333
Fritz Henglein, Konstantin Läufer

Modelagem de Esquemas em um Ambiente de Banco de Dados Orientado a Objetos 1049
Leonardo Chaib Rossetti, Ana Maria de C. Moura

Resúmenes 1325

REDES NEURALES CAPACITIVAS.

Autor: Pr. Eric Fimbel

Departamento de Computación y Tecnologia de la Información.

Universidad Simon Bolivar.

RESUMEN.

RESUMEN.

Hemos venido estudiando un tipo particular de redes: las redes de neuronas capacitivas.

Utilizan neuronas parecidas a condensadores que se cargan progresivamente y se descargan brutalmente.

Cada neurona tiene memoria de su pasado inmediato. Esa capacidad produce fenómenos temporales al nivel de la red completa: oscilaciones entre dos imágenes en caso de reconocimiento ambiguo, reconocimiento de secuencias temporales de imágenes, etc.

Por otra parte, utilizamos un sistema de crecimiento sináptico competitivo. Las sinapsis que llegan a una neurona tienen plasticidad, y compiten entre si durante su crecimiento.

Cada neurona tiene una cantidad constante de un cierto recurso, repartido entre las sinapsis que llegan hacia ella. La fuerza de una sinapsis crece en forma no lineal con la cantidad de recurso que tiene (satura). Por lo tanto, a diferencia del modelo de Weber, la fuerza sináptica total no queda constante.

Hemos comprobado que este tipo de aprendizaje es róbusto y bastante ráipido.

Para experimentar, utilizamos redes sencillas. Son matrices de hasta 16 x 16 pixels, totalmente conectadas, sin nodos escondidos. Son más lentas que las redes de Hopfield, pero tienen una capacidad de memorización y reconocimiento mucho mayor.

I INTRODUCCION.

Hemos venido estudiando un tipo particular de redes: las redes de neuronas capacitivas.

Utilizan neuronas parecidas a condensadores que se cargan progresivamente y se descargan brutalmente.

Cada neurona tiene memoria de su pasado inmediato. Esa capacidad produce fenómenos temporales al nivel de la red completa: oscilaciones entre dos imágenes en caso de reconocimiento ambiguo, reconocimiento de secuencias temporales de imágenes, etc.

Por otra parte, utilizamos un sistema de crecimiento sináptico competitivo. Las sinapsis que llegan a una neurona tienen plasticidad, y compiten entre si durante su crecimiento.

Cada neurona tiene una cantidad constante de un cierto recurso, repartido entre las sinapsis que llegan hacía ella. La fuerza de una sinapsis crece en forma no lineal con la cantidad de recurso que tiene (satura). Por lo tanto, a diferencia del modelo de Weber, la fuerza sináptica total no queda constante.

Hemos comprobado que este tipo de aprendizaje es róbusto y bastante rápido.

Para experimentar, utilizamos redes sencillas. Son matrices de hasta 16 x 16 pixels, totalmente conectadas, sin nodos escondidos. Son más lentas que las redes de Hopfield, pero tienen una capacidad de memorización y reconocimiento mucho mayor.

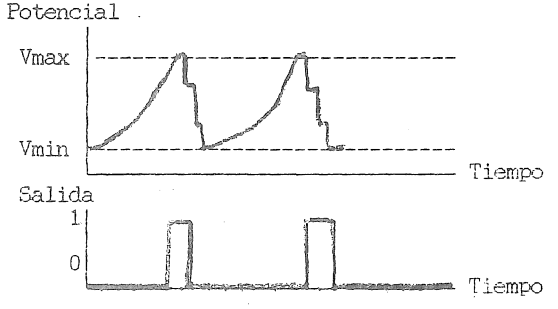
II. LAS NEURONAS CAPACITIVAS.

II.1 PRESENTACION.

Una neurona capacitiva tiene un POTENCIAL DE MEMBRANA V, que cambia con las SEÑALES excitatorias o inhibitorias que recibe.

Cuando el potencial pasa de un umbral Vmax, la neurona emite un POTENCIAL DE ACCION, o sea una señal hacia las demás neuronas.

El potencial de acción rebaja el potencial de la neurona hacia un valor Vmin, y el ciclo puede comenzar de nuevo.



La SALIDA de la neurona es un booleano que representa el potencial de acción. vale 1 cuando hay potencial de acción, 0 sino.

Se dice que la neurona está ACTIVA o PRENDIDA cuando su salida vale 1, APAGADA cuando vale 0.

El potencial de acción puede durar varios ciclos. Su duración es un parámetro de las neuronas.

Cuando no recibe señales, la neurona tiende a regresar a su POTENCIAL DE REPOSO, Vmin. Es un proceso más lento que la emisión del potencial de acción. Pasa desapercebido cuando la neurona está activa.

II.2 MODELO BIOLOGICO SUBYACENTE.

Las neuronas animales tienen un potencial de membrana, o POLARIZACION. Cuando están inactivas, este potencial es negativo (≈ -60 MiliVoltios).

La neurona tiende a regresar a su potencial de reposo, gracias a un sistema de bombeo que mantiene las concentraciones internas de iones K+ y Na+ constantes (alta concentración de K+, baja concentración de Na+).

Cuando recibe excitaciones por las sinapsis que le llegan, la neurona se DEPOLARIZA: su potencial de membrana aumenta (por ejemplo, sube desde -60 MiliVoltios hasta -40 MiliVoltios).

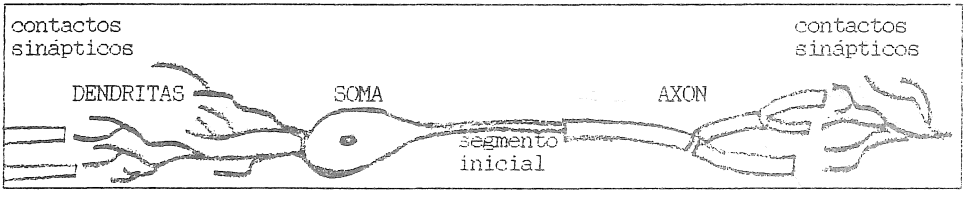
Al pasar el potencial de cierto umbral, se crea en la parte inicial del AXON (la "cola" de la neurona) un POTENCIAL DE ACCION SODICO.

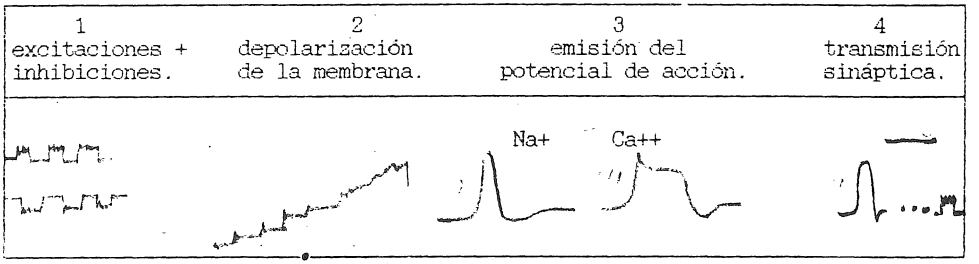
En el segmento inicial del AXON se produce una entrada brutal de iones Na+. Un poco más tarde, se produce una salida brutal de iones K+, y el potencial de la membrana vuelve a bajar (REPOLARIZACION).

El potencial de acción provocado por estos movimientos de iones se propaga en el axón, hacia las sinapsis. Allí, provoca el mando de una señal hacia las demás neuronas.

En los años setenta, se ha descubierto que las neuronas del sistema nervioso central podían emitir otro tipo de potencial de acción: el POTENCIAL DE ACCION CALCICO.

Este tipo de potencial, detectado anteriormente en ciertas fibras musculares, se debe a la entrada de iones Ca++ en vez de Na+. Presenta una duración mucho mayor que el potencial de acción sódico (de 10 a 30 milisegundos, versus 3 milisegundos).





II.3 LAS SINAPSIS.

Las sinapsis son los contactos que transmiten señales de una neurona hacia otra. tienen una FUERZA, que traduce la influencia de la señal sobre la neurona receptora. La fuerza sináptica puede variar durante los procesos de aprendizaje. eso explica la memorización de imagenes por parte de la red.

En nuestro modelo, existen sinapsis EXCITATORIAS e INHIBITORIAS. Desde la neurona i hacia la neurona j, existen dos conexiones:

EXC(i,j), excitatoria ; INH(i,j), inhibitoria.

Las sinapsis son DISIMETRICAS. Si queremos mandar una señal de vuelta desde j hacia i, tenemos que utilizar dos sinapsis más:

EXC(j,i), excitatoria ; INH(j,i), inhibitoria.

También, existen AUTOCONEXIONES: son sinapsis excitatorias e inhibitorias desde una neurona hacia ella misma:

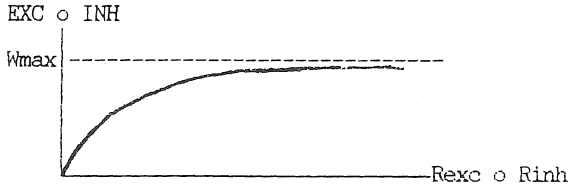
EXC(i,i), excitatoria ; INH(i,i), inhibitoria.

La fuerza sináptica resulta de un cierto RECURSO, presente en la neurona receptora. Cada sinapsis tiene una cierta cantidad de recurso, que determina su fuerza.

Notamos Rexc(i,j) el recurso que tiene la sinapsis EXC(i,j).
Notamos Rinh(i,j) el recurso que tiene la sinapsis INH(i,j).

La relación entre EXC() y Rexc() por una parte, y entre INH() y Rinh() por otra, es NO LINEAL. Para describirla, utilizamos una función sigmoïde W = σ(R):

W = 2 * Wmax / (1 + e ** (- G * R)) - Wmax



II.4 MODELO BIOLOGICO SUBYACENTE.

Las sinapsis son contactos entre dos neuronas. Generalmente hacen contacto entre el axón de la neurona emisora y una ESPINA DENDRITAL de la neurona receptora.

La TRANSMISION SINAPTICA se hace como sigue:

Al recibir un potencial de acción, la neurona emisora libera un NEUROTRANSMISOR en el ESPACIO SINAPTICO.

La membrana de la neurona receptora contiene RECEPTORES, que capturan el neurotransmisor liberado.

Al capturar el neurotransmisor, cada receptor deja pasar una microcorriente, excitatoria o inhibitoria, según su dirección.

En el sistema nervioso central, existen autoconexiones (neuronas cuyos axones extienden una ramificación hacia su propio cuerpo).

También, existen sinapsis excitatorias y otras inhibitorias, según sea el neurotransmisor utilizado(Por ejemplo, el glutamato es excitator y el Gaba es inhibitor).

Cada neurona puede mandar sinapsis inhibitorias Y excitatorias a la vez. En efecto, se detectó que muchas neuronas utilizan varios neurotransmisores.

Por otra parte, las sinapsis de los organismos superiores son generalmente disimetricas. Existen sinapsis bidireccionales simétricas, o asimétricas (con fuerzas distintas). De todos modos, se modelan facilmente mediante dos sinapsis, EXC(i,j) y EXC(j,i).

Por fín, la fuerza de una sinapsis depende basicamente de la cantidad de moléculas receptoras de neurotransmisor que contiene la membrana de la neurona receptora en el sitio de contacto.

En realidad, no sabemos si el recurso modela la concentración de receptores en la membrana o la concentración local de transcriptores del receptor (intervienen en su síntesis).

II.5 LAS ECUACIONES DE FUNCIONAMIENTO NEURAL: VERSION BASICA.

V(j,t) : Potencial de la neurona j al instante t.
S(j,t) : Salida de la neurona j al instante t. Es un booleano {0,1}.
EXC(i,j): Fuerza de la sinapsis excitatoria desde i hacia j.
INH(i,j): Fuerza de la sinapsis inhibitoria desde i hacia j.
E(j,t) : Señal total que recibe la neurona j al instante t.

Tenemos:

$$V(j,t+1) = V(j,t) * (1 - \beta) + E(j,t) - DV * S(j,t)$$
$$S(j,t+1) = (V(j,t+1) >= Vmax) * (1 - S(j,t)) + (V(j,t+1) >= DV) * S(j,t)$$
$$E(j,t) = \sum_i (EXC(i,j) - INH(i,j)) * S(i,t)$$

Interpretación:
Las sinapsis excitatorias e inhibitorias se comportan como una sola sinapsis de fuerza EXC(i,j)-INH(i,j). En cambio, durante el aprendizaje, ambas sinapsis crecen en forma independiente.

Cuando no recibe señales, el potencial tiende hacia 0, debido al coeficiente (1 - β). 0 es el potencial de reposo de las neuronas.

Cada vez que la salida está en 1, el potencial baja de DV

La salida se coloca en 1 si:
- El potencial pasa de Vmax o si
- La neurona ya estaba disparando (S(j,t) = 1) y
  su potencial permite seguir disparando (V(j,t+1) >= DV).

Estas ecuaciones representan una aproximación inexacta de una neurona animal, pero permiten una implantación fácil sin alejarse demasiado de su modelo inicial.

II.6 DIFERENCIA CON LOS MODELOS CLASICOS DE NEURONAS.

Clasicamente, se usan neuronas sin memoria. Por ejemplo, la neurona bipolar clásica se dispara según la ecuación siguiente:

S(j,t+1) = (E(j,t+1) >= 0)

En cambio, las neuronas capacitivas tienen memoria: el potencial V representa el resultado de las señales y disparos anteriores.

Observamos que el "grado de memoria" es ajustable. Si hacemos β=1, DV=0, Vmax=0 en las ecuaciones, tenemos neuronas sin memoria:

S(j,t+1) = (E(j,t+1) >= 0)

Por otra parte, las neuronas capacitivas tienen una caracteristica importante:

No hay forma de determinar el estado interno a partir de la salida, debido a la discontinuidad de la función que determina la salida a partir del estado interno.

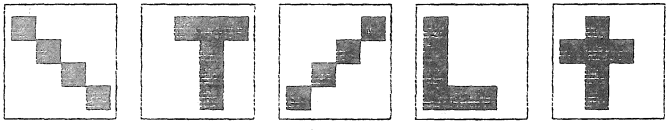
Nos parece que eso es la caracteristica genérica que provoca el comportamiento particular de las redes capacitivas.

III. FUNCIONAMIENTO DE LA RED.

La red se compone de una matriz de hasta 16x16 puntos. Cada punto está conectado con todos los demás. El ejemplo presentado a continuación utiliza imagenes de 4x4 puntos.

En un primer tiempo, presentamos a le red un conjunto de imagenes y la entrenamos para que las reconozca. Presentamos succesivamente cada imagen a la red.

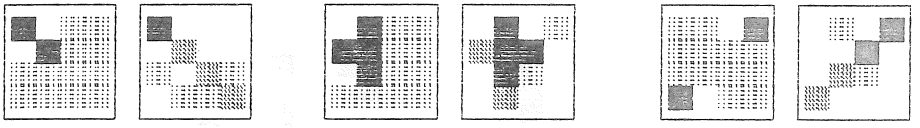
A cada presentación, se modifican las conexiones para grabar la configuración. Se repite el proceso 5 veces.



Luego, se da a la red una imágen parcial que deseamos que reconozca: la CONFIGURACION INICIAL. Los puntos tienen 3 estados posibles: Negro (prendido), Blanco (apagado) o Gris (indeterminado).

Dejamos correr la red, y vemos cuál es su ESTADO FINAL. En el estado final, cada punto tiene un tono de grís que describe su ACTIVIDAD (el promedio de sus salidas durante los 16 ciclos anteriores). Negro significa siempre prendido y blanco siempre apagado.

Estado inicial | Estado final | Estado inicial | Estado final | Estado inicial | Estado final



Observamos lo siguiente:

En el estado final, la imágen reconocida siempre aparece más oscura. Eso caracteriza un reconocimiento correcto.

Para detectarlo formalmente, deberiamos utilizar una neurona fuera de la imagen, que se activaria solamente cuando el reconocimiento es correcto.

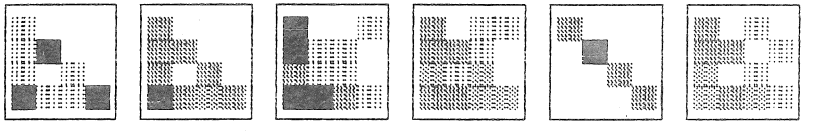
En los ejemplos, siempre hemos definido menos de 6 puntos (38% de la imagen). Eso basta para reconocer la imágen.

Además de la imágen reconocida, hay puntos parásitos que presentan una actividad (baja). Pertenecen casi siempre a imágenes que tienen puntos en común con la configuracion inicial.

Eso se debe a que las neuronas presentan una ACTIVIDAD ALEATORIA: pueden activarse solas (ver más adelante).

En el ejemplo anterior, Las configuraciones iniciales corresponden a una sola imagen. Veamos ahora lo que pasa si la configuración inicial es ambigua, o sea corresponde a más de una imágen. El ejemplo siguiente es una configuración ambigua, parte "L" y parte "\".

Estado inicial | L | \ | L | intermedia | \ | intermedia | ETC.



Observamos lo siguiente:

La red está OSCILANDO entre dos configuraciones: la "L" y la "\".

Generalmente, se considera que las oscilaciones se deben a las conexiones negativas. No es cierto. Las pruebas hechas sin conexiones negativas muestran también oscilaciones.

En ciertos casos, se estabiliza sobre una sola de las dos. Eso se debe a una mayor estabilidad de una de las dos configuraciones.

Las configuraciones más grandes son más estables, asi como las que no solapan con ninguna otra configuración aprendida.

Casi nunca se llega a ver las dos al mismo tiempo en forma estable. Eso se debe a las conexiones negativas: al aparecer una configuración aprendida, tiende a apagar los nodos de las demas.

Por fín, veamos lo que pasa cuando se deja la red totalmente libre. La configuración inicial tiene todos sus nodos indeterminados:

Estado inicial

ETC.

Observamos lo siguiente:

Debido al RUIDO TERMICO (la actividad aleatoria de las neuronas), la red muestra en forma aleatoria las configuraciones aprendidas.

Al dejar funcionar la red cierto tiempo, se nota que la frecuencia de aparición de las imagenes no es homógena.

Depende de la estabilidad propia de cada imágen. Las imagenes más grandes son más estables, asi como las que no solapan con ninguna otra.

IV. ADICIONES AL MODELO DE NEURONA.

IV.1 EL RUIDO TERMICO Y LA ENERGIA DE UNA NEURONA.

El ruido caracteriza la actividad aleatoria de cada neurona. Se modela en la forma siguiente:

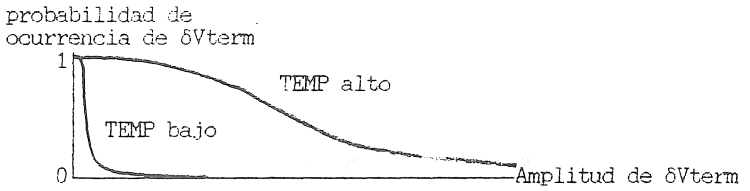
Cada neurona tiene excitaciones aleatorias de valor δVterm.

La probabilidad que occura la excitación δVterm baja a la medida que el AUMENTO DE ENERGIA DE LA NEURONA que provoca es más alto. La función de energia de la neurona se describe en anexo.

La probabilidad de que ocurra la excitación δVterm para la neurona j, al tiempo t, esta determinada mediante la ecuación:

Proba(δVterm) = e ** (- G * V(j,t) * δVterm / TEMP)

TEMP es la TEMPERATURA NEURAL de la red. Caracteriza la facilidad con la cuál las neuronas cambian solas de potencial. Es un parametro EXOGENO. Veamos su influencia:



Observamos lo siguiente:

Si TEMP = 0, no hay cambios aleatorios.

Si TEMP es muy alto, todos los cambios pueden ocurrir con una probabilidad cerca de 1, la red no puede estabilizarse.

La probabilidad de tener cambios es baja cuando el potencial de la neurona es alto. El ruido térmico REGULA EL POTENCIAL (y por lo tanto la actividad) de las neuronas.

IV.2 PRESENTACION DE UNA CONFIGURACION A LA RED.

Los puntos que están prendidos en la configuración inicial pueden terminar apagados o semi apagados en la configuración final: la red tiene capacidad de corregir la configuración presentada. Eso se debe al método que utilizamos para presentar una configuración a la red:

Cuando un nodo está prendido en la configuración, recibe una EXCITACION SUPLEMENTARIA de valor δVexc. Cuando está apagado, recibe una INHIBICION SUPLEMENTARIA de valor -δVinh.

Si δVexc y δVinh son grandes (>> Vmax) , entonces los nodos quedan en la configuración inicial. En cambio, si son pequeños, la red tiene mucha facilidad para "corregirlos".

Además, estas excitación e inhibición suplementarias no siempre se realizan. Por ejemplo, se hará solamente una vez cada 3 ciclos. En este caso, el PERIODO del refrescamiento es de 3.

Eso permite presentar a la red varias configuraciones en forma simultánea: presentamos cada una a su turno. Un periodo de 3 permite presentar 3 imágenes simultáneas.

En la práctica, un cambio de periodo no influye mucho sobre el reconocimiento. Si aumentamos el periodo, generalmente, se prenden y apagan nodos parásitos de vez en cuando.

IV.3 LAS ECUACIONES COMPLETAS DE FUNCIONAMIENTO DE LAS NEURONAS.

Describimos ahora las ecuaciones completas de funcionamiento, que integran ruido y presentación de imágenes.

Con las notaciones anteriores, tenemos ahora:

$$V(j,t+1) = V(j,t) * (1 - \beta) + E'(j,t) - DV * S(j,t)$$
$$S(j,t+1) = (V(j,t+1) >= Vmax) * (1 - S(j,t)) + (V(j,t+1) >= DV) * S(j,t)$$
$$E'(j,t) = E(j,t) + \delta Vterm(t) + \delta Vexc(j,t) - \delta Vinh(j,t)$$
$$E(j,t) = \sum_i (EXC(i,j) - INH(i,j)) * S(i,t)$$

δVexc(j,t) = δVexc ssi j está prendido en la configuración inicial y hay exposición en el ciclo t. sino, es nulo.

δVinh(j,t) = δVinh ssi j está apagado en la configuración inicial y hay exposición en el ciclo t. sino, es nulo.

V. EL APRENDIZAJE DE IMAGENES.

V.1 CONDICIONES DE CRECIMIENTO SINAPTICO.

Las fuerza de las sinapsis se modifica con la experiencia del sistema. Hay PLASTICIDAD SINAPTICA. Cuando se presenta una imágen nueva a la red, las sinapsis se modifican para grabarla.

Modelamos eso mediante una TEMPERATURA SINAPTICA. Lo más alta esta temperatura, lo más alta la probabilidad de que crezcan las sinapsis.

Cada categoria de sinapsis tiene su propia temperatura. Actualmente, las temperaturas son parámetros exógenos, mánejados por el programa:

Texc es la temperatura de las sinapsis excitatorias.
Tinh es la temperatura de las sinapsis inhibitorias.

La probabilidad de que una sinapsis cualquiera EXC(i,j) o INH(i,j) se ponga a crecer depende de varios factores:

La probabilidad crece con la actividad de la neurona i (promedio de sus últimas salidas).

La probabilidad es más alta si la sinapsis estuvo creciendo al ciclo anterior (inercia de crecimiento).

Además, si se trata de una sinapsis excitatoria, EXC(i,j):

La probabilidad crece con la temperatura sináptica Texc.

La probabilidad CRECE con el potencial V(j).

La probabilidad baja con la cantidad de recurso Rexc(i,j).

Si se trata de una sinapsis inhibitoria, INH(i,j) :

La probabilidad crece con la temperatura sináptica Tinh.

La probabilidad DECRECE con el potencial V(j).

La probabilidad baja con la cantidad de recurso Rinh(i,j).

Para escoger las sinapsis que crecen, utilizamos un método de Monte Carlo presentado en anexo.

V.2 EL MODELO BIOLOGICO SUBYACENTE.

Estamos modelando un tipo de crecimiento sináptico que no pasa de ser una hipótesis. Se basa en tres ideas:

Los receptores de un neurotransmisor están atraidos por el neurotransmisor o una substancia liberada con el neurotransmisor.

Lo más activa está una sinapsis, lo más alta está la densidad promedia del neurotransmisor o la substancia en el espacio sináptico.

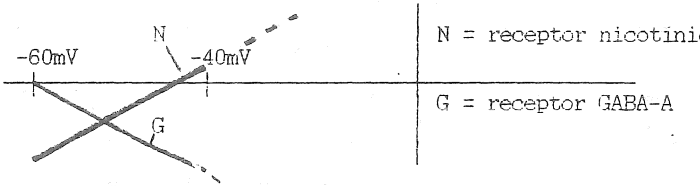
La probabilidad de atraer más receptores decrece con la densidad de receptores ya presentes.

Eso corresponde a una saturación, por cualquier razón que sea: repulsiones entre receptores, etc.

Los receptores migran más facilmente en la membrana hacia las sinapsis más activas, si no están trabajando.

El receptor de un neurotransmisor inhibitor (por ejemplo, el GABA-A) trabaja poco si el potencial de membrana esta bajo (la corriente que atraviesa el receptor es casi nula).

El receptor de un neurotransmisor excitator (por ejemplo, el receptor nicotinico de la acetilcolina) trabaja poco si el potencial de membrana esta alto (la corriente que atraviesa el receptor es casi nula).



V.3 EL APRENDIZAJE COMPETITIVO.

La fuerza sináptica depende de la cantidad de recurso concentrado por la sinapsis. El recurso existe en cantidades limitadas en las neuronas: por lo tanto las sinapsis compiten para conseguirlo.

Cada categoría de sinapsis, positiva y negativa, utiliza un recurso distinto. Las sinapsis de una catagoria compiten entre sí, y no interfieren con las sinapsis de la otra categoria.

En lo que sigue, consideramos las sinapsis excitatorias. El caso de las inhibitorias es idéntico.

Cada neurona tiene una cantidad fija de recurso: R(j).

Cada sinapsis hacia j tiene una cantidad de recurso, R(i,j,t), que depende del tiempo t.

El recurso que no pertenece a ninguna sinapsis está LIBRE. es Rlib(j,t), que también depende del tiempo. Tenemos, a cada momento:

$$R(j) = Rlib(j,t) + \sum_i R(i,j,t)$$

En un principio, las sinapsis tienen poco recurso. $R(i,j,0) \approx 0$, y queda mucho recurso libre, $Rlib(j,0) \approx R(j)$. Si la sinapsis (i,j) crece, lo hace mediante la captura de un poco de recurso libre:

$$R(i,j,t+1) = R(i,j,t) + \delta R$$
$$Rlib(j,t+1) = Rlib(j,t) - \delta R$$

Después de la etapa inicial de crecimiento, no queda recurso libre. Rlib(i,j,t) ≈ 0. Cuando una sinapsis (i,j) crece, debe quitar recurso a otra sinapsis que llega a la misma neurona, (k,j). HAY COMPETENCIA:

$$R(i,j,t+1) = R(i,j,t) + \delta R$$
$$R(k,j,t+1) = R(k,j,t) - \delta R$$

La sinapsis (k,j) se escoge en forma aleatoria. Pero, en este sorteo, las sinapsis débiles tiene mayor probabilidad de ser victimas que las sinapsis fuertes.

La implantación de este sorteo se hace mediante un algoritmo de Monte Carlo presentado en anexo.

V.4 DIFERENCIAS Y PARECIDOS CON LOS MODELOS CLASICOS DE APRENDIZAJE.

Los dos modelos más conocidos de aprendizaje son los de HEBB y WEBER. El modelo de Hebb dice que deben crecer las sinapsis entre nodos i y j conjuntamente prendidos en la configuración a aprender.

Esta estrategia minimiza la energía de la red para la imágen que está aprendiendo. Eso es cierto solamente para las redes sin autoconexiones, y con conexiones simétricas.

Nuestro criterio de crecimiento sináptico es una generalización del criterio de Hebb: la sinapsis excitatoria EXC(i,j) crece con mayor probabilidad si i está activa y el potencial de j es alto.

Si i está activa, tiene una alta probabilidad de estar prendida (la actividad es el promedio de los últimos estados).

Por otra parte, Si el potencial V(j,t) es alto, hay una fuerte probabilidad para que j se prenda pronto.

Las conexiones inhibitorias no están contempladas por el criterio de Hebb.

El modelo de WEBER dice que la sumatoria de todas las sinapsis que llegan a un nodo debe quedarse constante:

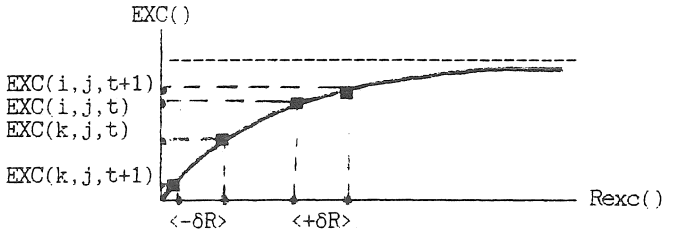
$$\sum_i W(i,j,t) = W(j) \text{ constante.}$$

Este criterio introduce la competencia entre sinapsis, pero directamente al nivel de la fuerza sináptica.

Por ejemplo, en caso de que W(i,j) crezca:

$W(i,j,t+1) = W(i,j,t) + \delta W$ debe acompañarse de:
$W(k,j,t+1) = W(k,j,t) - \delta W$

En cambio, la competencia para el recurso no mantiene constante la sumatoria de las fuerzas sinápticas. Por ejemplo, si la sinapsis (i,j) aumenta su fuerza, y quita recurso a la sinapsis (k,j):



V.5 EL MODELO BIOLOGICO SUBYACENTE.

El RECURSO es la cantidad de moléculas receptoras del neurotransmisor en la membrana de la neurona receptora (puede ser el receptor mismo, o un transcriptor del receptor).

La junción sináptica que nos sirve de modelo para el aprendizaje competitivo es la junción nervio motor vs. célula muscular. Su desarrollo epigenético ha sido bastante estudiado.

> En una primera etapa, llega un axón de motoneurona cerca de la fibra muscular. Establece una primera junción sináptica. Por debajo de esta junción, comienza a concentrarse el receptor (nicotinico) que hasta el momento estaba uniformemente diseminado en toda la membrana de la fibra muscular.
>
> Llegan más y más axones de motoneuronas (hasta 30). Debido a un tropismo, van directamente hacia la primera sinapsis, y aqui establecen su sinapsis. En esta etapa, crecen sin competir.
>
> Debido a una actividad motor automática (los fetos se mueven), las sinapsis trabajan. En esta etapa, compiten.
>
> Al final de esta etapa, queda una sola: la más fuerte. Las demás están completamente eliminadas. La que queda no siempre es la primera: es la más fuerte.

Se supone, sin haberlo comprobado hasta el momento, que en el sistema nervioso central occure también el fenómeno de competencia sináptica. Esta competencia está por supuesto limitada a las sinapsis que utilizan un mismo neurotransmisor. Ahora, ¿ en que medida influye la topografía compleja de una neurona sobre este fenómeno ?

V.6 DETERMINACION DE LA CANTIDAD DE RECURSO INICIAL.

La cantidad de recurso inicial es un parámetro sumamente importante para el funcionamiento global de la red. Para la aplicación de reconocimiento de imágenes:

> Si existe poco recurso, pocas sinapsis podrán crecer. Por lo tanto, el número y tamaño de las imagenes que se pueden aprender son muy limitados.
>
> Si existe demasiado recurso, no hay competencia sináptica. Durante el reconocimiento, demasiados nodos tienden a prenderse, la red no discrimina bién entre las imágenes.

V.7 DETERMINACION DE LAS TEMPERATURAS SINAPTICAS.

Como hemos dicho anteriormente, en el protótipo actual, las temperaturas son parámetros exógenos. Son muy importantes: ya que definen los momentos durante los cuales la red aprende, cambiar completamente su comportamiento. Presentamos aquí dos heurísticas, que corresponden a dos versiones distintas del protótipo.

Aprendizaje separado de la corrida:

Los modos APRENDIZAJE y CORRIDA están separados. En el modo APRENDIZAJE, las temperaturas sinápticas Texc y Tinh esatán en alto, la red puede aprender. Durante la CORRIDA, están en 0.

Aprendizaje de asociaciones temporales (imágen A ,luego imágen B):

Se aprende A y B por separado como visto en el caso anterior.

Luego, se presenta A y se deja estabilizar la red.

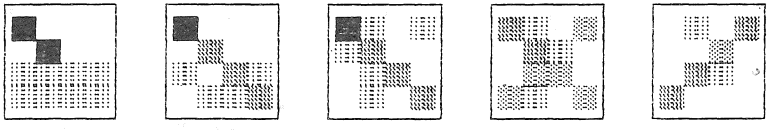
Luego, se presenta B, y se pone en alto la temperatura de las sinapsis excitatorias, Texc, durante pocos ciclos.

Crecen las conexiones excitatorias entre los nodos de A y los de B: la imágen A tiende a prender la imágen B.

Veamos un ejemplo de reconocimiento de asociación: '\',luego '/'.

Se configura una parte de la imagen inicial (50 %).
Después de C ciclos, se libera completamente la imágen.
La segunda imagen aparece superpuesta con la primera.
La segunda elimina la primera y queda sola.

Estado inicial | Reconoce la '\' | Liberamos todos nodos. | Superpone '\' y '/' | Configura la '/'.



VI. CONCLUSIONES PROVISIONALES.

Las redes capacitivas parecen tener una alta capacidad de almacenamiento (hasta 8 patrones en una red de 4 x 4 puntos), asi como un buén nivel de reconocimiento (a veces, basta prender un solo nodo para reconocer una imágen completa).

Los algoritmos de aprendizaje són bastante robustos: el resultado es independiente del orden de presentación de las imagenes, el número de pasadas necesarias es bajo (nunca ha necesitado más de 6 pasadas).

Por fín, el aprendizaje de asociaciones temporales (A luego B) es uno de nuestros mayores logros: sin nodos ni conexiones suplementarios, con un algoritmo sencillo (el mismo algoritmo de aprendizaje sirve), la red reconoce asociaciones temporales es forma bastante buena.

En cambio, si la convergencia es rápida, los tiempos de cálculo son mayores a los de las redes de hopfield. Hace falta optimizar los algoritmos y estructuras de datos utilizados

Además, por el momento, la red no tiene sistema de salida: el ojo humano vee perfectamente si la red ha reconocido una imágen o no, pero todavia no hemos implantado un sistema automático para hacerlo.

AGRADECIMIENTOS.

A J.F. ARIAS, C. CONTRERAS, G. GONZALEZ, G. HERNANDEZ DE VIVAS, P.L. ROMERO, V. YRIARTE, y a todos los que han participado a los desarrollos de programas, los experimentos y el establecimiento de los modelos de redes capacitivas.

BIBLIOGRAFIA.

ABU MUSTAFA Y.S.
"Connectivity versus entropy", Neural information processing, Denver, PP.1.8, 1988.
ACKLEY D.H. , HINTON G. E., SEJNOWSKY T.J.
"A learning algorithm for Boltzmann machines" Cognitive science V.9, pp. 147-160, 1985.
BORG- GRAHAM L.J.
"Simulations suggest information processing roles for the divers currents in hippocampal neurons", Neural information processing, Denver, FP.1.8, 1988.
GUEVARA M., GLASS L., MACKEY M. SCHRIER A.
"Chaos in neurobiology", IEEE FP.790 798, 1983.
HAMMOND C., TRITSCH D.
"Neurobiologie cellulaire, Doin Editeur, 1990
HUBEL D.H. , WIESEL T.N.
"Sequence regularity and geometry of orientation columns in the monkey striata cortex" Journal of comparative Neurology 158. PP. 267-274, 1974.
KLARSFELD A.
"La compartimentation des fibres musculaires", La recherche v. 228, pp. 116-119, 1991.
KOSKO B.
"Bidirectional associative memories" IEEE Ttransactions on systems, vol 18 FP.49-60, 1988.
LANG K.J. , HINTON G.E.
"The development of the time delay Neural Netword Arqchitecture for speech recognition" Tech.Rep. CMUCS, pp. 152, Carnegie Mellon University, 1988.
LAUFER R. CHANGEUX J.P.
"Activity dependant regulation of gene expression in muscles and neuronal cells" Mol. Neurobiol , t3, pp. 1-53, 1989.
LAUTRUP B.
"Monte Carlo Methods in theoretical high-energy physics", ACM, V. 28, PP. 358 362., 1985.
NAKAMURA K., ICHIKAWA A.
"Timing information in transcient behavior of neuropopulations", IEEE transactions on systems, v.19, pp 1188-1192, 1989.
VON DER MALSBURG C.
"Self organization of sensitive cells in the striata cortex" Kibernetik vol 14. PP 85-100, 1973.

VII. ANEXO.

VII.1 ECUACIONES DEL RUIDO TERMICO.

La variación de energía debida a una excitación se aproxima mediante el siguiente modelo:

Consideramos la neurona como un condensador. En este caso, tenemos la siguiente relación entre su potencial V(j,t) y su carga eléctrica Q(j,t) :

V(j,t) = Q(j,t) / Capacidad(j)

En primera aproximación consideramos una excitación como una entrada de carga δQterm que provocaría una variación de potencial δVterm si la neurona estuviese aislada.

δVterm = δQterm / Capacidad(j)

La energia a gastar para realizar esta variación de carga es:

δEnergia = V(j,t) * δQterm

δEnergia (δVterm) = V(j,t) * δVterm * Cste.

La probabilidad de aceptar la excitación δVterm es dada por la distribuición de Boltzmann:

Proba(δVterm) = e ** (- G * δEnergia (δVterm) / Temp)

VII.2 EL METODO DE MONTE CARLO.

Se utiliza en varios procesos: el ruido térmico, la competencia sináptica, etc. Permite tomar una decisión con una probabilidad dada. El problema no es simple: una probabilidad se establece generalmente a partir de un gran número de experimentos. El método general es el siguiente:

Calcular la probabilidad PRB de tomas una decisión.

Escoger un número UMB, al azar en [0..1], con una ley uniforme.

Si PRB >= UMB se acepta la decisión.
En el caso contrario, se la rechaza.

VII.3 EL RUIDO TERMICO.

Excitación aleatoria de la neurona j, al tiempo t:

Escogemos al azar un δVterm.

Calculamos P = Probabilidad (δVterm).

Escogemos un número UMB, al azar en [0..1], con una distribuición uniforme.

Si P >= UMB, se realiza la excitación.
En caso contrario, no pasa nada.

VII.4 CRECIMIENTO SINAPTICO.

Se escoge las sinapsis EXC(i,j) que van a crecer en la forma siguiente:

Se chequean todos los nodos i tales que actividad(i) >= 0.7*16 (16 es el máximo de las actividades).

Escogemos un nodo j al azar. Si V(j,t) < Vmax*0.7, no hace nada. Vmax es el valor máximo del potencial de membrana.

Calculamos PRB = actividad(i) * V(j,t) * (Rmax-Rexc(i,j)) * Texc
Rmax es el valor máximo del recurso de una sinapsis.

Si EXC(i,j) estaba creciendo al tiempo t-1, PRB = PRB*2.

Normalizamos PRB en [0..1].

Escogemos un número UMB, al azar en [0..1], con una distribuición uniforme.

Si PRB > UMB, se aumenta la sinapsis EXC(i,j).
Sino, no pasa nada.

Las sinapsis inhibitorias se manejas en la misma forma, salvo que:

Se descarta el nodo j si V(j) > 0.3*Vmax.

PRB = actividad(i) * (Vmax - V(j,t)) * (Rmax-Rexc(i,j)) * Tinh

VII.5 COMPETENCIA SINAPTICA.

Dado el nodo j, ya hemos escogido una sinapsis (i,j) que va a aumentar su fuerza. debemos escoger una sinapsis (k,j) que sea la víctima, o sea tal que le quitamos su recurso.

Escogemos al azar un nodo k.

Normalizamos EXC(k,j) en [0..1]. llamemos EXC el valor normalizado.

Escogemos un número UMB, al azar en [0..1], con una distribuición uniforme.

Si EXC < UMB, se realiza la excitación.
En caso contrario, se itera el proceso de búsqueda de víctima.

On Formally Refining Timed Petri Nets*

Sergio Bandinelli†
Dipartimento di Elettronica
Politecnico di Milano

Esteban Feuerstein‡
Dipartimento di Informatica e Sistemistica
Universita di Roma "La Sapienza"
on leave from ESLAI

Abstract

Petri nets —and some extended models based on them— have been recognized as an appropiate formalism for describing concurrent and time-dependent systems, because of its combination of a graphical and mathematically founded model. However, Timed Petri nets lack a formal, satisfactory and widely accepted notion of refinement. In this paper we propose a formal definition of refinement for a powerful, highly expressive and general time-oriented formalism, called ER-nets (a high level extension of the original Petri net model). We describe refinements as net morphisms that allow us to relate two net specifications at different levels of abstraction. Then we show how this definition may be used to define decomposition schemata for ER-nets, that guarantee that the decomposed net is actually a refinament of the original one. In particular, we provide nondeterministic, sequential and parallel decompositions. The use of these transformations is ilustrated through examples.

*This work has been done while both authors were at ESLAI (Escuela Superior Latino Americana de Informatica).

†Supported by Engineering S.p.A.

‡Partially supported by Fundacion Antorchas - Argentina.

1 Introduction

Petri nets are increasingly being used as a modeling language for representing concurrent systems. This is not only a consequence of their neat graphical representation, but also because the Petri net formalism [Reisig 82] is a well-known mathematical model, supported by theoretical studies. Moreover, many methods have been proposed for analyzing system properties specified with Petri nets; for example [Ghezzi 89b] proposes symbolic execution and [Berthomieu 83] presents time analysis. In addition, some extensions to the model for representing time dependent systems, i.e. systems which correctness depend on real time constraints (such as process control systems or communicating protocols) have been proposed, widening the area of application of Petri nets.

The ability of modelling a system hierarchicaly is a valuable and necessary feature of any specification formalism. But one of the main drawbacks of Petri nets is the unavailabity of a satisfactory hierarchical decomposition facility [Harel 86]. The main aim of this work is to provide a formal way to hierarchicaly decompose a net.

There have been many efforts to provide hierarchical developement for Petri nets. A very flexible way of relating system descriptions at different levels of abstraction, consisting in mapping single transitions into whole computations, is provided by [Meseguer 88] (for place transition nets) and by [Feuerstein 88] (for a simple model of Petri nets with time). A proposal towards defining refinements for TB-nets (a timed basic model described in [Ghezzi 89]) is discussed in [Felder 88] and a set of rules and restrictions to correctly expand a transition in a full process are given.

In this paper we adopt the net formalism proposed in [Morasca 89] and [Ghezzi 91] called Environment/Relation nets (ER-nets). An ER-net is a high level Petri net model, which may be particularly useful in the modelling of timming issues and has the property of being general and basic enough to include previous models, defined as ad hoc time extensions of the Petri net formalism.

We follow the ideas from [Meseguer 88] and [Feuerstein 88] to give formal notion of refinement for ER-nets. However, the definition does not provide a mechanism to correctly refine a net. Therefore we provide some topological transformations on ER-nets, based in the ones proposed in [Morasca 89], that guarantee that the net obtained after the tranformation is indeed an implementation of the original one, according to the formal definition of refinement. These transformations may be used by the designer during a hierarchical developement of the system specification.

This work is structured as follows. Section 2 presents the basic definitions of ER-nets as given in [Morasca 89]. In section 3, ER-nets are defined in terms of graphs and a notion of equivalence between nets is given. A notion of refinements for the ER-net model is formally given in section 4. Section 5 is dedicated to the description of topological transformations on ER-nets and are ilustrated through an example. The final conclusions are mentioned in section 6.

2 Basic definitions

Environment Relation nets (ER-net for short) are a high-level extension of Petri nets, similar in scope to Predicate Transition nets [Genrich 86] and Coloured nets [Jensen 86]. In this paper, we assume that the reader knows the basics of Petri nets and their intended

uses, therefore we limit ourselves to providing the basic definitions of the ER-net model and refer the reader to [Reisig 82] [Peterson 77] for basic definitions and notation relative to the Petri net formalism.

In the ER-nets model, tokens carry information in the form of environments, i.e. as a mapping from identifiers to values. The enabling of a transition depends not only on the presence of environments in the input places, but also on the number and on the information that those environments contain. The firing of a transition has the effect of destroying the environments enabling that firing and creating a new tuple of environments in the output places of the transition. The link between the removed tuple and the produced one is described by a relation associated with the transition. More than one output tuple may be related with an input tuple, giving a source of nondeterminism in the behaviour of the transition. Note that the relations also specify the number of input and output environments that participate of a firing, for each input and output place. That is, a relation links multisets of environments in the input places to multisets of environments in the output places.

2.1 Formal definition of ER-nets

Definition 2.1 (ER-net) *An ER-net is a 6-tuple* (P, T, E, F, R, M_0) *where* P, *the set of places,* T, *the set of transitions and* E, *the set of environments are arbitrary sets such that* $P \cap T = \emptyset; P \cap E = \emptyset; T \cap E = \emptyset$
$P \cup T \neq \emptyset$ *and* $E \neq \emptyset$
The flow relation F *links places to transitions and transitions to places and verifies that:* $F \subseteq (P \times T) \cup (T \times P)$ *and* $\Pi_1(F) \cup \Pi_2(F) = P \cup T$ *(i.e. there are not isolated places or transitions)*
If $E^{\oplus}$ *denotes the set of all finite multisets (also called bags) of elements of* E, *and* $A \rightarrow E^{\oplus}$, *the set of all aplications from a set* $A \subseteq P$ *to* $E^{\oplus}$, *then the relation* R *may be defined for each transition* $t \in T$ *as:*
$R(t) \subseteq ((^{\bullet}t \rightarrow E^{\oplus}) \times (t^{\bullet} \rightarrow E^{\oplus}))$
Finally, the initial marking M_0 *belongs to the set of all possible markings of the net:*

$$M_0 \in (P \rightarrow E^{\oplus})$$

During the evolution of an ER-net, the relation $R(t)$ links the tuples of environments removed from the places of the preset of t (denoted by $^{\bullet}t$) with the tuples of environments produced in the places of the postset of t (denoted by $t^{\bullet}$). The domain of the relation, $\Pi_1(R(t))$ is called the set of the enabling tuples of transition t. The range of the relation, $\Pi_2(R(t))$, is the set of tuples which can be produced by the firing of transition t.

In this work we restrict ourselves to the case where, presets, postsets and markings are finite, even in the presence of an infinite number of places and transitions, and although the set of possible markings were infinite. This kind of restriction is also present in [Degano 89] and we believe it is not a significant inconvenience.

Definition 2.2 (Set of Firings) *Let* $N = (P, T, E, F, R, M_0)$ *be an ER-net. The set of firings of* N *is defined as:*

$$Y = \{\langle t, en, pr\rangle / t \in T, \langle en, pr\rangle \in R(t)\}$$

For each firing $y \in Y$, en is called the *enabling tuple* (of bags) and pr, the *produced tuple* (of bags) for transition t. A firing $y = \langle t, en, pr \rangle$ is said to be *enabled* in the marking m iff $\forall p \in {}^\bullet t\,(en(p) \subseteq m(p))$.

The effect of the firing of a transition, called *firing occurrence* is a modification of the marking of the net in which the firing is enabled. The marking after the firing is equal to the marking before the firing except for the input places of the fired transition, where the enabling tuple is substracted, and for the output places, where the produced tuple is added.

2.2 Monotonic ER-nets

A time-dependent system is characterized by a monotonic behaviour with respect to time. This means that the output of an action may not be produced at a time which is greater than the time the inputs where created. To model this fact using ER- nets, a (partial) ordering between environments is introduced. We interpretate that an environment is less than other environment if it was produced before. An ER-net that models a monotonic behaviour must satisfy that, for each transition, the associated relation is monotonic, i.e. no environment belonging to a produced tuple is less than any environment of the corresponding removed tuple. Formaly,

Definition 2.3 (Monotonic Relation) *Let $\geq$ be a reflexive and transitive relation defined on the elements of E, i.e. let $(E, \geq)$ be a partial order, and let A and B be two arbitrary sets. A relation*

$$R \subseteq ((A \rightarrow E^{\oplus}) \times (B \rightarrow E^{\oplus}))$$

is said to be a monotonic relation w. r. t. $(E, \geq)$ iff

$$\forall \langle m, m' \rangle \in R\,(\forall a \in A\, \forall b \in B(\forall e \in m(a)\, \forall e' \in m'(b)\,(e' \geq e)))$$

Definition 2.4 (Monotonic ER-net) *An ER-net is said to be monotonic w. r. t. $(E, \geq)$ iff*

1. *$(E, \geq)$ is a partial order and*
2. *$\forall t \in T\,(R(t)$ is monotonic w. r. t. $(E, \geq)$*

Monotonic ER-nets support specification of control flow, data and functionalities, including time issues. Control flows are represented by the flow of environments; data are represented as information associated with environments and functionalities are specified as relations associated with transitions. The ordering defined on the set of environments allows also the representation of time issues. The following example shows how this can be done.

2.3 An Example

We show how a cake recipie may be specified using ER-nets. Making a cake is a real time process, since its correctness depends on time constraints (otherwise you would eat a burnt cake or a raw one!). Here follows the informal enuntiation of the recipie, as it may be found in a cooking book.

The ingredients for making a cake are: 4 eggs, one pound of flower, one pound of sugar and one ounce of butter. The preparation of the dough takes 10 or 15 minutes and the baking time is of 40 minutes. Beat the eggs for 5 minutes using an electric beater (or for 10 minutes if you use a fork). Mix the flower, the sugar and the butter and then add the beaten eggs. Put the preparation in the oven and that's all[1].

We may represent the recipie by an ER-net $\mathcal{N} = (P, T, E, F, R, M_0)$. The ingredients, the cake and the intermediate products are represented with environments having two pieces of information: the name of what they represent and the time at which it was produced. We write $e = (id, t)$ meaning that environment e represents id and was created at time t. The times associated with each environment provide us with a partial order[2] on the set E in a natual way: environment e is greater or equal than environment e' if e's time is greater or equal than e''s time.

Formally, the parameters of $\mathcal{N}$ are $P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$. $T = \{$Start, Mix, BtFrk, BtBtr, Add, Bake$\}$[3]. $E = \{(egg, t_e), (flower, t_f), (butter, t_b), (sugar, t_s), (cake, t_c), (beateneggs, t_{be}), (mixture, t_m), (dough, t_d)\}$ where $t_e, t_f, t_b, t_s, t_c, t_{be}, t_m, t_d \in \mathbb{R}$. F is given by Figure 1; finally, the relations associated with the transitions and the initial marking are:

$$
\begin{aligned}
R(Start) &= \{[4(p_1,(egg,t_e)) \oplus (p_1,(flower,t_f)) \oplus (p_1,(butter,t_b)) \oplus (p_1,(sugar,t_s)),\\
&\quad (p_2,(flower,t_1)) \oplus (p_2,(butter,t_1)) \oplus (p_2,(sugar,t_1)) \oplus 4(p_3,(egg,t_1))]\}\\
&\quad where\ t_1 = Max(t_e,t_f,t_b,t_s)\\
R(Mix) &= \{[(p_2,(flower,t_f)) \oplus (p_2,(butter,t_b)) \oplus (p_2,(sugar,t_s)),(p_4,(mixture,t_2))]\}\\
&\quad where\ t_2 = Max(t_f,t_b,t_s) + 5\\
R(BtFrk) &= \{[4(p_3,(egg,t_e)),(p_5,(beateneggs,t_e+10))]\}\\
R(BtBtr) &= \{[4(p_3,(egg,t_e)),(p_5,(beateneggs,t_e+5))]\}\\
R(Add) &= \{[(p_4,(mixture,t_m)) \oplus (p_5,(beateneggs,t_{be})),(p_6,(dough,Max(t_m,t_{be})))]\}\\
R(Bake) &= \{[(p_6,(dough,t_d)),(p_7,(cake,t_d+40))]\}\\
M_0 &= 4(p_1,(egg,0)) \oplus (p_1,(flower,0)) \oplus (p_1,(butter,0)) \oplus (p_1,(sugar,0))
\end{aligned}
$$

□

3 Equivalence of ER-nets

In this section we see an ER-net as a ordinary directed graph. This approach allows us to define ER-net morphisms as a certain kind of graph morphisms and give a notion of

[1] The authors shall not be liable for damages caused by the use of this recipie.

[2] Actually it is a total order.

[3] BtFrk stands for "Beat with Fork" and BtBtr stands for "Beat with (electric) Beater".

equivalence of ER-nets based on the existence of a bijective morphism between the corresponding graphs. The same definitions may be appied to monotonic ER- nets, provided that the morphism respect the monotonicity of the relations.

3.1 ER-nets as graphs

We can associate a graph with an ER-net in the following way. The nodes of the graph correspond to all the possible markings of the net (the reachable and unreachable ones) and the oriented arcs to the firings. The initial marking is a distinguished node in the graph. The functions ∂_0 and ∂_1, called *source* and *target* respectively, are defined for each firing $\langle t, en, pr \rangle \in Y$ as $\partial_0 = \Pi_2$ and $\partial_1 = \Pi_3$.

Definition 3.1 (ER-graph) *Given the ER-net* $\mathcal{N} = (P, T, E, F, R, M_0)$ *the corresponding ER-graph is defined as the graph*

$$\mathcal{G}[\mathcal{N}] = (\partial_0, \partial_1 : Y \longrightarrow (P - E^{\oplus}), M_0)$$

There is an important remark to be made with respect to the structure of the set of nodes. The set of nodes of a net is isomorphic to the free conmutative monoid generated by the pairs (p, e) with $p \in P$ and $e \in E$. In symbols [4]:

$$(P - E^{\oplus}) \simeq (P \times E)^{\oplus}$$

Therefore, there is a monoid structure defined on the nodes of the graph. A generic element $m \in (P \times E)^{\oplus}$ may be written as the formal sum: $m = \bigoplus_{i,j} k_{i,j}(p_i, e_j)$ where the order of the sumands is superfluous. This notation means that $k_{i,j}$ copies of environment e_j are present in place p_i. Addition is defined in $(P \times E)^{\oplus}$ for two elements m and m' as follows

$$m \oplus m' = \bigoplus_{i,j} k_{i,j}(p_i, e_j) \oplus \bigoplus_{i,j} k'_{i,j}(p_i, e_j) = \bigoplus_{i,j} (k_{i,j} + k'_{i,j})(p_i, e_j)$$

resulting 0 (the empty bag) as the neutral element.

The previous discusion and the fact that it is easy to translate from one notation to the other[5] allows us to use alternatively any of the two isomorphic structures.

Notation: We adopt the traditional notation from graph theory $\alpha : m \longrightarrow m'$ meaning that for the arc α, $\partial_0(\alpha) = m$ and $\partial_1(\alpha) = m'$

Observe that there is no missing information in the definition of ER-graph, since the original ER-net can be easily recovered from the graph.

Example.
Figure 2 is a graphical representation[6] of the ER-graph corresponding to the ER-net $\mathcal{N}$ of our running example. □

[4] $E^{\oplus}$ may be seen as a function from E to the natural numbers, indicating the number of instances in the bag for each environment. Then $(P - E^{\oplus}) \simeq (P - E - \mathcal{N}) \simeq (P \times E) - \mathcal{N} \simeq (P \times E)^{\oplus}$

[5] Since $(P - E^{\oplus})$ and $(P \times E)^{\oplus}$ are isomorphic, there exists a bijection $\mathcal{F} : (P - E^{\oplus}) \longrightarrow (P \times E)^{\oplus}$ defined in the following way: let $m \in (P \times E)^{\oplus}$; $m' \in (P - E^{\oplus})$. If $m = \bigoplus_{i,j} k_{i,j}(p_i, e_j)$ and $m' = \mathcal{F}(m)$

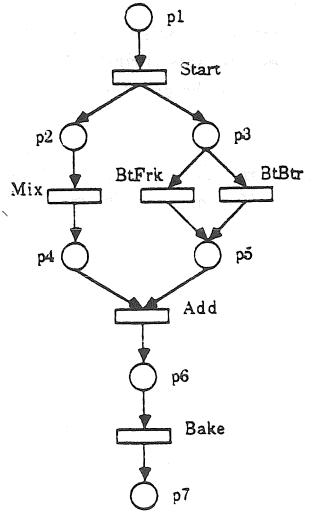


Figure 1: Graphical representation of ER-net $\mathcal{N}$.

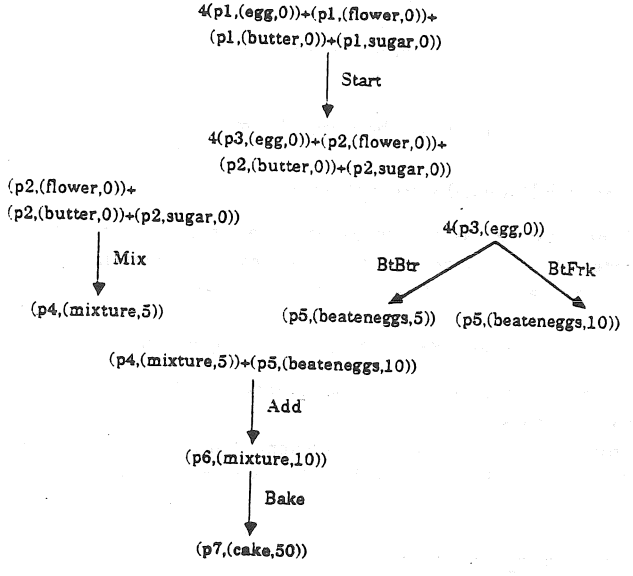


Figure 2: Graphical representation of the ER-graph $\mathcal{G}[\mathcal{N}]$.

3.2 ER-nets morphisms and equivalence

Two ER-graphs can be related through a morphism satisfying certain restrictions. The morphism respectively maps the nodes and arcs (firings and markings) of one net to the nodes and arcs of another net, and respects the graph structure and the monoid structure of the set of nodes.

Definition 3.2 (ER-graph morphism) *Let $\mathcal{N} = (P, T, E, F, R, M_0)$ and $\mathcal{N}' = (P', T', E', F', R', M_0')$ be ER-nets. An ER-graph morphism $h : \mathcal{G}[\mathcal{N}] \longrightarrow \mathcal{G}[\mathcal{N}']$ is a pair of functions $h = \langle f, g \rangle$ such that*
$f : Y \longrightarrow Y'$
$g : (P \times E)^{\oplus} \longrightarrow (P' \times E')^{\dot{+}}$
and
$g \circ \partial_0 = \partial_0' \circ f$
$g \circ \partial_1 = \partial_1' \circ f$
g is a monoid morphism, i.e. satisfies
$\forall m_1, m_2 \in (P' \times E')^{\oplus}\ g(m_1 \oplus m_2) = g(m_1) \dot{+} g(m_2)$ *and* $g(0) = 0$
g preserves the initial marking, i.e., $g(M_0) = M_0'$.

Now we are ready to provide a notion of equivalence between ER- nets.

Definition 3.3 (ER-nets equivalence) *Two ER-nets $\mathcal{N} = (P, T, E, F, R, M_0)$ and $\mathcal{N}' = (P', T', E', F', R', M_0')$ are said to be equivalent, we write $E \equiv E'$, iff there exists a ER-graph morphism $h : \mathcal{G}[\mathcal{N}] \longrightarrow \mathcal{G}[\mathcal{N}']$ where $h = \langle f, g \rangle$ such that*
$f : Y \longrightarrow Y'$ *is bijective*
$g : (P \times E)^{\oplus} \longrightarrow (P' \times E')^{\dot{+}}$ *is a monoid isomorphism*

Clearly '$\equiv$' is an equivalence relation between ER-nets.

3.3 Monotonic ER-graphs morphisms

Monotonic ER-nets have an associated partial order defined on the set of environments. This partial order $(E, \geq)$ may be naturally extended to the markings of an ER-net. The result is another partial order $((P \longrightarrow E^{\oplus}), \succeq)$ defined as follows. If $m, m' \in (P \longrightarrow E^{\oplus})$ then

$$m \succeq m' \Longleftrightarrow \forall p \in P(\forall e \in m(p), \forall e' \in m'(p)(e \geq e'))$$

In other words, each environment belonging to marking m has to be greater or equal than each environment of marking m'.

The monotonicity property may now be enuntiated for ER- graphs. We say that an ER-graph $\mathcal{G}[N] = (\partial_0, \partial_1 : Y \longrightarrow (P \longrightarrow E^{\oplus}), M_0)$ is monotonic iff

$$\forall y \in Y(\partial_1(y) \succeq \partial_0(y))$$

---

then $\forall i (1 \leq i \leq n) m'(p_i) = \bigoplus_j k_{i,j} e_j$.

[6]The figure is necessarily an incomplete representation, since the graph has an infinite number of nodes and arcs.

That is, we require all the environments belonging to the produced tuple to be grater or equal than the each environment of the corresponding removed tuple. Note that all the enviroments involved in a firing must be related by $\succeq$.

Definition 3.4 (Monotonic ER-graph morphism) *We call monotonic ER-graph morphism an ER-graph morphism $h = \langle f, g \rangle$ that in addition verifies that*

$$\forall m, m' \in (P - E^{\oplus})\, m \succeq m' \Rightarrow g(m) \succeq g(m')$$

4 Refinemets for ER-nets

Here we provide a formal notion of refinement for ER-nets and monotonic ER-nets. We begin by defining the ER-Computation graph, which contains as arcs all the possible computations of an ER-net. Then, a refinement of an ER-net is defined as an implementation morphism relating its ER-graph with its ER-Computation graph. In this way, a complete computation may be associated with a single transition via an implementation morphism. Finally, we observe that the initial definition of refinement as an implementation morphism appears to be too permisive: thus we propose a restricted version of the notion of refinement, that we call proper refinement.

4.1 ER-Computation graph

A computation is a single firing, a sequence of computations or a set of parallel computations. The computation graph is a graph whose nodes stand for markings and whose arcs represent all possible computations of an ER-net. For technical reasons, we include in the set of arcs an idle (identity) computation for each marking, meaning that the environments are left in their places.

Definition 4.1 (ER-Computation graph) *The ER-Computation graph corresponding to the ER-net $N = (P, T, E, F, R, M_0)$ is the graph*

$$C[N] = (\partial_0, \partial_1 : C \longrightarrow (P - E^{\oplus}), M_0)$$

The following rules define the arcs of the graph:

$$\frac{\langle t, en, pr \rangle : en - pr \in Y}{\langle t, en, pr \rangle : en - pr \in C}$$

$$\frac{m \in (P - E^{\oplus})}{m : m - m \in C}$$

$$\frac{\alpha : m_1 - m_1' \in C \text{ and } \beta : m_2 - m_2' \in C}{\alpha \oplus \beta : m_1 \oplus m_2 - m_1' \ominus m_2' \in C}$$

$$\frac{\alpha : m - m' \in C \text{ and } \beta : m' - m'' \in C}{\alpha ; \beta : m - m'' \in C}$$

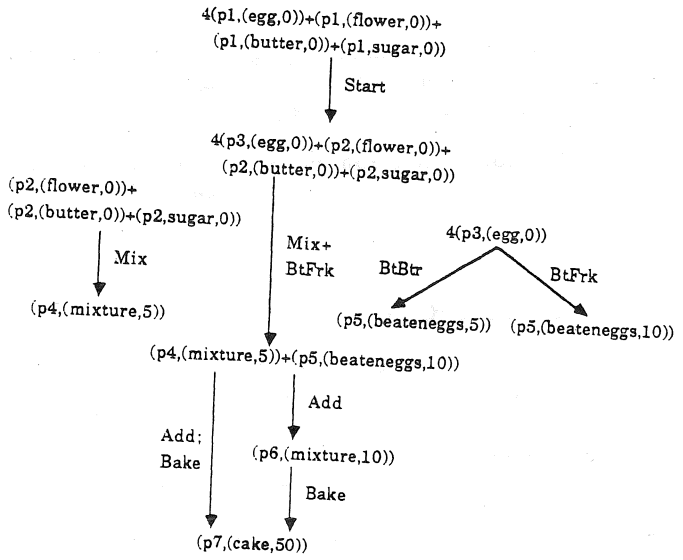


Figure 3: Graphical representation of the ER-computation graph $\mathcal{C}[N]$.

The definition introduces two operations between computations: '$\ominus$' and ';' that we call respectively parallel and sequential composition of computations[7]. With the above definition, there are some different arcs in the computation graph, that intuitively describe the same computation. Thus, we identify these arcs by completing the definition with the following axioms: Let $\alpha, \alpha', \beta, \beta', \gamma \in C$

- $(\alpha \ominus \beta) \ominus \gamma = \alpha \ominus (\beta \ominus \gamma)$
- $\alpha \oplus \beta = \beta \oplus \alpha$
- $\alpha \ominus 0 = \alpha$
- $\partial_0(\alpha); \alpha = \alpha; \partial_1(\alpha) = \alpha$
- $(\alpha; \beta); \gamma = \alpha; (\beta; \gamma)$
- $(\alpha \oplus \alpha'); (\beta \oplus \beta') = (\alpha; \beta) \ominus (\alpha'; \beta')$

As a consecuence of the definition we have that $(C, \oplus, 0)$is a conmutative monoid[8].

Example

Figure 3 shows the computation graph $\mathcal{C}[N]$. for net N, describing the cake recipie[9] □

[7] We use the same simbol $\oplus$ to denote the formal sum between markings and the parallel composition of computations. However this should cause no confusion, since it is always clear from the context to which operation we are making reference.

[8] Here 0 represents the empty transition that goes from the empty marking to the empty marking.

[9] Once again, the figure is necessarily an incomplete representation, since the graph has an infinite number of nodes and arcs.

4.2 Implementation morphisms and refinements

If $\mathcal{N}$ and $\mathcal{N}'$ are two ER-nets, an implementation morphism gives us the possibility of mapping a transition of $\mathcal{N}$ into an entire computation of $\mathcal{N}'$, which may contain many sequential and parallel steps. To achieve this capability it is enough to restrict our attention to the ER-graph of $\mathcal{N}$ and the ER-Computation graph of $\mathcal{N}'$, (the latter may also be seen as an ER-graph where the firings are computations of $\mathcal{N}'$).

Definition 4.2 (Implementation morphism) *Let $\mathcal{N}$ and $\mathcal{N}'$ be ER-nets. An implementation morphism from $\mathcal{N}$ to $\mathcal{N}'$ is an ER-graph morphism $h : \mathcal{G}[\mathcal{N}] \rightarrow \mathcal{C}[\mathcal{N}']$.*

Definition 4.3 (Refinement of an ER-net) *Let $\mathcal{N}$ and $\mathcal{N}'$ be ER-nets. We say that $\mathcal{N}'$ is a refinement of $\mathcal{N}$ iff there exists an implementation morphism from $\mathcal{N}$ to $\mathcal{N}'$.*

4.3 Proper Refinements for ER-nets

The definition of refinement given so far does not force the refined net to behave always as the original ER-net. The refined net may contain computations that are not image of any transition in the original net. Our conclusion is that this first general notion of refinement is too permisive. Therefore, we provide here a more restrictive notion, that we call proper refinement, requiring that every computation in the refined net corresponds to the image of a firing of the original net, or it can be completed with other computations in shuch a way that the obtained computation stisfies the previous condition.

Definition 4.4 (Proper Refinement of ER-nets) *Let $\mathcal{N}$ and $\mathcal{N}'$ be ER-nets. We say that $\mathcal{N}'$ is a proper refinement of $\mathcal{N}$ iff there exist an implementation morphism $h = \langle f, g \rangle : \mathcal{G}[\mathcal{N}] \rightarrow \mathcal{C}[\mathcal{N}']$ and for every computation η' in $\mathcal{N}'$, there exist computations α', β' and γ' such that*

$$\alpha' ; (\eta' \doteqdot \beta') ; \gamma = f^{\$}(\alpha)$$

where α is a computation of $\mathcal{N}$.

Notation. $h^{\$} = \langle f^{\$}, g^{\$} \rangle : \mathcal{C}[\mathcal{N}] \rightarrow \mathcal{C}[\mathcal{N}']$. is the homomorphic extension of $h = \langle f, g \rangle : \mathcal{G}[\mathcal{N}] \rightarrow \mathcal{C}[\mathcal{N}']$.

4.4 Monotonic computation graphs and refinements

The notion of computation graph may be adequated to monotonic ER-nets by restricting ourselves to monotonic computations, i.e. those that verify the monotonicity property as stated above (see section 3.3).

Definition 4.5 (Monotonic ER-Computation graph) *Given the monotonic ER-net $\mathcal{N} = (P, T, E, F, R, M_0)$, its monotonic ER-Computation graph is defined as the graph*

$$\mathcal{M}[N] = (\partial_0, \partial_1 : MC \longrightarrow (P - E^{\oplus}), M_0)$$

The set MC of monotonic computations is the set of arcs of the graph and is defined by the following rules. The rules remain the same of definition 4.1 except for the parallel composition rule where an additional condition is needed.

$$\frac{(\langle t, en, pr\rangle : en \multimap pr) \in Y}{\langle t, en, pr\rangle : en \multimap pr \in MC}$$

$$\frac{m \in (P - E)^{\oplus}}{m : m \multimap m \in MC}$$

$$\frac{\alpha : m_1 \multimap m_1' \in MC \text{ and } \beta : m_2 \multimap m_2' \in MC \text{ and } m_1' \oplus n}{\alpha \oplus \beta : m_1 \oplus m_2 \multimap m_1' \oplus m_2' \in MC} \qquad \underline{\quad \supset m_2}$$

$$\frac{\alpha : m \multimap m' \in MC \text{ and } \beta : m' \multimap m'' \in MC}{\alpha ; \beta : m \multimap m'' \in MC}$$

As before, some computations (arcs) are identified by the axioms of section 4.1.

In this case $(C, \oplus, 0)$ is not necessarily a monoid, the $\oplus$ operation is not more total since it is not always defined for every pair of elements of MC. Note that the obtained monotonic ER-Computation graph $\mathcal{M}[\mathcal{N}]$ may also be seen as a monotonic ER-graph, where the arcs are the computations of $\mathcal{N}$.

When refining monotonic ER-nets we would like to ensure that the monotonicity properties are maintained. This is achieved by requiring the implementation morphism to be a monotonic morphism.

Definition 4.6 (Monotonic Implementation morphism) *Let $\mathcal{N}$ and $\mathcal{N}'$ be monotonic ER-nets. A monotonic implementation morphism from $\mathcal{N}$ to $\mathcal{N}'$ is a monotonic ER-graph morphism $h : \mathcal{G}[\mathcal{N}] \multimap \mathcal{M}[\mathcal{N}']$.*

Definition 4.7 (Monotonicity-preserving refinement of an ER-net) *Let $\mathcal{N}$ and $\mathcal{N}'$ be monotonic ER-nets. We say that $\mathcal{N}'$ is a monotonicity-preserving refinement of $\mathcal{N}$ iff there exist a monotonic implementation morphism from $\mathcal{N}$ to $\mathcal{N}'$.*

Definition 4.8 (Monotonicity-preserving proper refinement of ER-nets) *Let $\mathcal{N}$ and $\mathcal{N}'$ be monotonic ER-nets. We say that $\mathcal{N}'$ is a monotonicity preserving proper refinement of $\mathcal{N}$ iff it is proper refinement and, at the same time, a momotonicity preserving refinement.*

5 Topological Transformations

ER-net specifications can be hierarchicaly developed applying the definition of refinement given in the previous sections. Every step of refinement relates two nets at different levels of abstraction via an implementation morphism.

During the design of a system, it ussualy happens that some refinement schemata are repeatedly used. In these cases, it becames tedious to verify everytime that the refined net is actually related to the original net by an implementation morphism. Thus, we propose some transformation schemata for ER-nets, which impose some contraints on the transformed net, but, on the other hand, guarantee that the obtained net is a refinement of the original one.

Here we consider the following three topological transformations for ER-nets: (the first two have been already proposed in [Morasca 89]).

- Nondeterministic decomposition
- Sequential decomposition
- Parallel decomposition

We will show that after applying nondeterministic decomposition to a net, an equivalent net is obtained, while sequential and parallel decomposition lead to a net which is a proper refinement of the original one. We are not claiming that this set of transformations is enough for describing all possible refinement schemata. We rather believe in an open environment where the designer has the possibility to define or tailor his own transformation schemata.

5.1 Nondeterministic decomposition

The nondeterministic decomposition allows the designer to make explicit the nondeteminism present in the relation associated with a transition by dividing the relation in a set of more "specific" ones and by associating each of these relations to a different new transition (see figure 4). The nondeterministic decomposition is in fact similar to the split operation as defined in [Morasca 89].

Definition 5.1 (Nondeterministic Decomposition) *Let* $\mathcal{N} = (P, T, E, F, R, M_0)$, *be an ER-net; let* $t \in T$ *and* $\{\mathcal{R}_{t,i}\}\ i \in I$ *be such that*

$$\forall i \in I\,(\mathcal{R}_{t,i} \subseteq R(t))\ and\ \bigcup_{i \in I} \mathcal{R}_{t,i} \subseteq R(t)\ and\ \bigcup_{i \in I} \mathcal{R}_{t,i} = R(t)$$

i.e. it is a family of subsets of $R(t)$ *covering* $R(t)$. *The net* $\mathcal{N}' = (P', T', E', F', R', M_0')$ *is obtained from* $\mathcal{N}$ *by nondeterministic decomposition of* t *under the set* $\{\mathcal{R}_{t,i}\}$ *iff:*

$$\begin{aligned}
P' &= P \\
T' &= (T - \{t\}) \cup \{split(t,i)/i \in I\} \\
E' &= E \\
F' &= F - \{\langle a,b\rangle / \langle a,b\rangle \in F\ and\ ((a = t)\ or\ (b = t))\} \cup \\
&\quad \{\langle a, split(t,i)\rangle / a \in {}^\bullet t\ i \in I\} \cup \{\langle split(t,i), b\rangle / b \in t^\bullet\ i \in I\} \\
R' &\quad such\ that\ \forall t' \in (T - \{t\}) R'(t') = R(t') \\
&\quad and\ R'(split(t,i)) = \mathcal{R}_{t,i} \\
M_0' &= M_0
\end{aligned}$$

Theorem 5.1 *Let* $\mathcal{N}$ *be an ER-net and* $\mathcal{N}'$ *obtained from* $\mathcal{N}$ *using nondeterministic decomposition (split) of transition* $t \in T$. *Then,* $\mathcal{N} \equiv \mathcal{N}'$.

Proof: The ER-graph structure is not modified by the nondeterministic decomposition (at most there is a change of names in the firings that involve the splitted transition). Therefore, there is an obvious isomorphism between $\mathcal{G}[\mathcal{N}]$ and $\mathcal{G}[\mathcal{N}']$. □

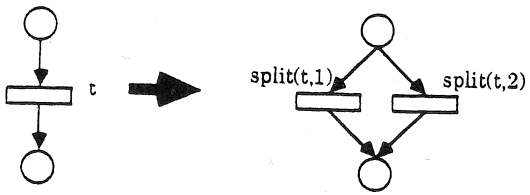


Figure 4: Nondeterministic Decomposition

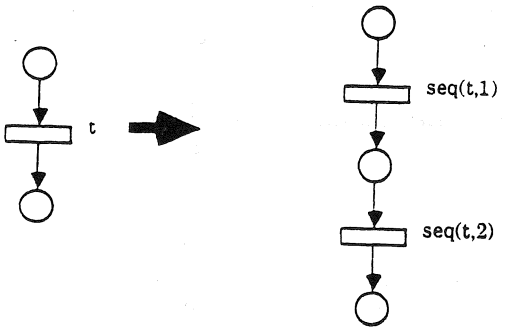


Figure 5: Sequential Decomposition

5.2 Sequential Decomposition

The intuitive idea of a sequential decomposition is to replace a transition with a sequence of two transitions (the sequence can be easily extended to more than two transitions). A set of new places is needed to connect the new transitions.

If the refined transition is $t \in T$, we will call the new ones $seq(t,1)$ and $seq(t,2)$ (see figure 5). Now the main question is what are the relations associated with these new transitions. It seems natural to request that the composition of the two new relations gives as result the relation originally associated with t. However, in order to obtain a proper refinament from a sequential decomposition, we need stronger conditions on the relations associated with the new transitions. These conditions are stated in the following restrictive version of the composition of relations.

Definition 5.2 (Restricted Composition of Relations) *Let P_i, E_i $i = 1 \ldots 3$ be ar-*

bitary sets of places and environments respectively and let $R_1 \subseteq (P_1 \times E_1)^{\oplus} \times (P_3 \times E_3)^{\ominus}$ *and* $R_2 \subseteq (P_3 \times E_3)^{\oplus} \times (P_2 \times E_2)^{\oplus}$ *be two relations. We say that* R *is the restricted composition of* R_1 *and* R_2 *and we write* $R = R_1 \odot R_2$ *iff*

1. $R = R_1 \circ R_2$
2. $\Pi_2(R_1) = \Pi_1(R_2)$. *We will call this set the set of Intermediate Markings (IM).*
3. *If the set of intermediate markings is* $IM = \{m_1, m_2, \ldots, m_i, \ldots\}$ *then* $m_j \in IM$ *cannot be written in terms of the rest of the intermediate markings* $IM - \{m_j\}$. *In other words,* $m_j = \bigoplus_i k_i m_i \Longrightarrow k_i = 0$ *for* $i \neq j$ *and* $k_j = 1$

Intuitively, the last requirement avoids the possibility of computations in the refined net that may have no counterpart in the original net. Now we are ready to define a topological transformation based on the sequential decomposition of a transition. The formal construction is the following:

Definition 5.3 (Sequential Decomposition) *Let* $N = (P, T, E, F, R, M_0)$ *be an ER-net and* $t \in T$ *a transition. The net* $N' = (P', T', E', F', R', M_0')$ *is obtained from* N *by sequential decomposition of* t *iff:*

$$
\begin{array}{lcl}
P' & = & P \cup P'' \\
T' & = & (T - \{t\}) \cup \{seq(t,1), seq(t,2)\} \\
E' & = & E \cup E'' \\
F' & = & F - \{\langle a,b\rangle / \langle a,b\rangle \in F \; and \; ((a = t) \; or \; (b = t))\} \cup \\
 & & \{\langle a, seq(t,1)\rangle / a \in {}^{\bullet}t\} \cup \{\langle seq(t,1), b\rangle / b \in P''\} \cup \\
 & & \{\langle b, seq(t,2)\rangle / b \in P''\} \cup \{\langle seq(t,2), c\rangle / c \in t^{\bullet}\} \\
R' & & \textit{such that } \forall t' \in (T - \{t\}) R'(t') = R(t') \\
 & & \textit{and } R'(seq(t,1)) = \mathcal{R}_{t,1} \; R'(seq(t,2)) = \mathcal{R}_{t,2} \textit{ where } \mathcal{R}_{t,1} \odot \mathcal{R}_{t,2} = R(t) \\
M_0' & & \textit{such that } \forall p \in P M_0'(p) = M_0(p) \textit{and} \forall p \in P'' M_0'(p) = 0
\end{array}
$$

P'' is a new set of places and E'' is a new set of environments.

Note that the net N', obtained by sequential decomposition of transition t from the ER-net N, is also an ER-net.

The following lemma is used in the proof of theorem 6.2

Lemma 5.1 *Let N' be an ER-net obtained from another ER-net N, by sequential decomposition of transition t. Let m be a reachable marking restricted to the set of new places, i.e., $m \in (P'' \times E')^{\oplus}$. Then m can be written as a linear combination of intermediate markings.*

Proof: Since the only way an environment may appear in a place $p \in P''$ is by the firing of transition $seq(t,1)$, the resulting marking is a combination of the set $\Pi_2(R(seq(t,1))$, which is equal to the set of intermediate markings IM. In symbols:

$$m = \bigoplus_i k_i m_i$$

where the k_i are natural numbers and the $m_i \in IM$. □

Theorem 5.2 *Let $\mathcal{N}$ be an ER-net and $\mathcal{N}'$ obtained from $\mathcal{N}$ by sequentially decomposing transition $t \in T$. Then, $\mathcal{N}'$ is a proper refinement of $\mathcal{N}$. If, in addition, $\mathcal{N}$ is a monotonic ER-net and the relations $\mathcal{R}_{t,1}$ and $\mathcal{R}_{t,2}$ of the above definition are monotonic, then $\mathcal{N}'$ is a monotonicity preserving proper refinement of $\mathcal{N}$.*

Proof: We first show an (obvious) implementation morphism $h = \langle f, g \rangle$ between $\mathcal{N}$ and $\mathcal{N}'$. This morphism maps all firings of $\mathcal{N}$ to the corresponding firings of $\mathcal{N}'$, except for those where transition t is involved, which are mapped to the sequential composition of two suitable firings of $\mathcal{N}'$. Formally[10]:

$$\forall m \in (P \times E)^{\oplus}\ g(m) = m$$

$$\forall \langle t', en, pr \rangle \in Y \text{ such that } t' \in T - \{t\}\ f(\langle t', en, pr \rangle) = \langle t', en, pr \rangle$$

$$\forall \langle t, en, pr \rangle \in Y\ f(\langle t, en, pr \rangle) = \langle seq(t,1), en, m \rangle; \langle seq(t,2), m, pr \rangle \text{ where } m \in IM$$

In order to see that this is a proper refinement we have to verify that each computation of $\mathcal{N}'$ is part of a computation which is the homomorphic image of some other computation in $\mathcal{N}$. Let $\eta' : m \multimap m'$ be a computation of $\mathcal{N}'$. We consider the following cases:

1. Neither m nor m' involve new places P''.
2. The marking m of $\mathcal{N}'$ has a corresponding marking in $\mathcal{N}$, but m' does not (it involves places from P'').
3. The marking m' of $\mathcal{N}'$ has a corresponding marking in $\mathcal{N}$, but m does not (it involves places from P'').
4. Both m and m' involve places of P'', thus they have no corresponding marking in $\mathcal{N}'$ via the implementation homomorphism.

The first is the trivial case. The defined morphism assigns $g(m) = m$ and $g(m') = m'$, so the computation $\alpha : m \multimap m'$ is such that $f^{\$}(\alpha) = \eta'$. The other three cases are more interesting. Here we will only consider case 2, since the remaining two require a similar proof.

We have to show that there exist computations α', β' and γ' such that $\alpha'; \eta' \oplus \beta'; \gamma' = f^{\$}(\alpha)$. We can write m' as the union (sum) of a marking belonging to $(P \times E)^{\oplus}$ and a marking belonging to $(P'' \times E')^{\oplus}$ in the following way: $m' = m_P \oplus m_{P''}$. Now, applying lemma 6.1, $m_{P''}$ can be written as $m_{P''} = \bigoplus_i k_i m_i$.

Let r_i be a tuple in $R'(seq(t,2))$ such that $r_i = (en_i, pr_i)$ and $en_i = m_i$ (it always exists, because $IM = \Pi_1(R(seq(t,2)))$). Then the computations:

$$\begin{aligned} \alpha' &= m : m \multimap m \\ \beta' &= 0 : 0 \multimap 0 \text{ and} \\ \gamma' &= m_P; (k_1\langle seq(t,2), r_1 \rangle \oplus \ldots \oplus k_n\langle seq(t,2), r_n \rangle) : m' \multimap m'' \end{aligned}$$

where $m'' = m_P \oplus k_1 pr_1 \oplus \ldots \oplus k_n pr_n$, are such that $\alpha'; \eta' \oplus \beta'; \gamma' = f^{\$}(\alpha)$ and α is a computation of $\mathcal{N}$.□

[10] Note that if $m \in (P \times E)^{\oplus}$ then $m \in (P' \times E')^{\oplus}$

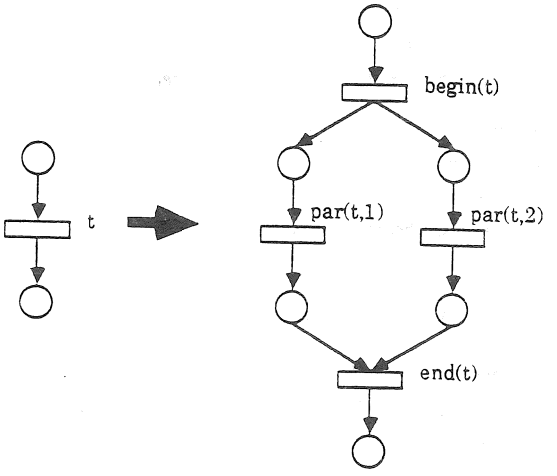


Figure 6: Parallel Decomposition

5.3 Parallel Decomposition

The intuitive idea of the parallel decomposition is to replace a transition by a set of 'more primitive' transitions that all together, carried out in parallel, produce the same effect of the original transition. In order to follow the requirements so as the decomposed net is a proper refinement of the given one, we include a transition at the beginnig and one at the end of the parallel decomposition (as shown in figure 6). Again the main question to be answered is what are the requirements that the relations associated with the new transitions have to accomplish. We begin by showing that the relations of the $par(t, i)$ transitions may be composed in one big relation:

Definition 5.4 (Parallel composition of relations) *Let P_i, E_i $i = 1 \ldots 4$ be arbitrary disjoint sets of places and environments respectively and let $R_1 \subseteq (P_1 \times E_1)^{\oplus} \times (P_2 \times E_2)^{\ominus}$ and $R_2 \subseteq (P_3 \times E_3)^{\oplus} \times (P_4 \times E_4)^{\ominus}$ be two relations. We say that R is the parallel composition of R_1 and R_2 and we write $R = R_1 \parallel R_2$ iff $R \subseteq (P_1 \cup P_3 \times E_1 \cup E_3)^{\oplus} \times (P_2 \cup P_4 \times E_2 \cup E_4)^{\ominus}$ and*

$$\begin{aligned} R \;=\; & \{(m_1 \oplus m_3, m_2 \ominus m_4)/ \\ & m_1 \in (P_1 \times E_1)^{\oplus}\, m_2 \in (P_2 \times E_2)^{\ominus}\, m_3 \in (P_3 \times E_3)^{\oplus}\, m_4 \in (P_4 \times E_4)^{\ominus} \\ & and\, (m_1, m_2) \in R_1\, (m_3, m_4) \in R_2\} \end{aligned}$$

Definition 5.5 (Parallel Decomposition) *Let N=(P, T, E, F, R, M_0) an ER-net and $t \in T$ a transition. A new net ER-net N'=(P', T', E', F', R', M_0') is obtained by parallel decomposition of N with the following contruction: Let $I = \{1 \ldots n\}$ a set of indexes and the families of sets indexed by I*

$$T_i \;=\; \{par(t, i)/i \in I\}$$

$$
\begin{aligned}
P_i &= \{inplace(t,i,j)/i \in I, j = 1,\ldots,m_i,\ m_i \in \mathcal{N}\} \cup \\
&\quad \{outplace(t,i,j)/i \in I, j = 1,\ldots,n_i,\ n_i \in \mathcal{N}\} \\
F_i &= \{\langle a,b\rangle/a = inplace(t,i,j),\ b = par(t,i),\ j = 1,\ldots,m_i\} \cup \\
&\quad \{\langle a,b\rangle/a = par(t,i),\ b = outplace(t,i,j),\ j = 1,\ldots,n_i\}
\end{aligned}
$$

Now we define $\mathcal{N}'$ in the following way:

$$
\begin{aligned}
P' &= P \cup (\bigcup_{i\in I} P_i) \\
T' &= (T - \{t\}) \cup \{begin(t), end(t)\} \cup (\bigcup_{i\in I} T_i) \\
E' &= E \cup E'' \\
F' &= F - \{\langle a,b\rangle/\langle a,b\rangle \in F \ and \ ((a = t) or (b = t))\} \cup (\bigcup_{i\in I} F_i) \cup \\
&\quad \{\langle a, begin(t)\rangle/a \in {}^\bullet t\} \cup \{\langle end(t), b\rangle/b \in t^\bullet\} \cup \\
&\quad \{\langle begin(t), b\rangle/b \in {}^\bullet par(t,i), i \in I\} \cup \{\langle a, end(t)\rangle/a \in par(t,i)^\bullet, i \in I\} \\
R' &\quad such\ that\ \forall t' \in T - \{t\}\, R'(t') = R(t')\ and \\
&\quad R'(begin(t)),\ R'(end(t)),\ R'(par(t,i))\, i \in I\ are\ such\ that \\
&\quad R'(begin(t)) \oplus (\|_{i\in I}\ R'(par(t,i))) \oplus R'(end(t)) = R(t) \\
M_0' &\quad such\ that\ \forall p \in P\ M_0'(p) = M_0(p)\ and\ \forall p \in \bigcup_{i\in I} P_i,\ M_0'(p) = 0
\end{aligned}
$$

E'' is a new set of environments.

Theorem 5.3 *Let $\mathcal{N}$ be an ER-net and $\mathcal{N}'$ obtained from $\mathcal{N}$ by a parallel decomposition of transition $t \in T$. Then, $\mathcal{N}'$ is a proper refinement of $\mathcal{N}$. If, in addition, $\mathcal{N}$ is a monotonic ER-net and the relations $R'(begin(t))$, $R'(end(t))$ and $R'(par(t,i))$ of the above definition are monotonic, then $\mathcal{N}'$ is a monotonicity preserving proper refinement of $\mathcal{N}$.*

Proof: We will only show the implementation morphism, since the rest of the proof is analogous to the one for sequential decomposition. The implementation morphism $h = \langle f, g\rangle$ between $\mathcal{N}$ and $\mathcal{N}'$ is given by:

$$\forall m \in (P \times E)^{\oplus}\ g(m) = m$$

$$\forall \langle t', en, pr\rangle \in Y\ such\ that\ t' \in T - \{t\}\ f(\langle t', en, pr\rangle) = \langle t', en, pr\rangle$$

$$
\begin{aligned}
&\forall \langle t, en, pr\rangle \in Y\ f(\langle t, en, pr\rangle) = \\
&\langle begin(t), en, m\rangle; (\|_i\ \langle par(t,i), m_i, m_i'\rangle); \langle end(t), m', pr\rangle
\end{aligned}
$$

□

5.4 An Example

Here we show how the cake recipie, presented in section 2.2 may be hierarchicaly developed, via a sequence of formal refinements. We begin with the ER-net of Figure 7, where the realation associated with the transition is[11]:

[11] MkCk stands for MakeCake, and PrDgh, for Prepare Dough.

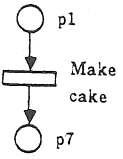


Figure 7: A high level specification of the recipie.

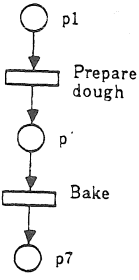


Figure 8: The recipie after sequencial decomposition.

$$R(MkCk) = \sim\{[4(p_1,(egg,t_e)) \ominus (p_1,(flower,t_f)) \ominus (p_1,(butter,t_b)) \ominus (p_1,(sugar,t_s)),$$
$$(p_7,(cake,t_c))]\} \; where \; t_c = Max(t_e,t_f,t_b,t_s+\alpha \; and \; \alpha \in \{50,55\})$$

Actually, the recipie may be seen as composed of two sequencial steps. The first step involves the preparation of the dough, while the second one is baking. Therefore, it seems natural to apply sequential decomposition to obtain the net of Figure 8, with the following associated relations.

$$R(PrDgh) = \{[4(p_1,(egg,t_e)) \ominus (p_1,(flower,t_f)) \ominus (p_1,(butter,t_b)) \ominus (p_1,(sugar,t_s)),$$
$$(p',(dough,t_d))]\} \; where \; t_d = Max(t_e,t_f,t_b,t_s+\alpha \; and \; \alpha \in \{10,15\})$$
$$R(Bake) = \{[(p',(dough,t_d)),(p_7,(cake,t_d+40))]\}$$

Now we concentrate on the preparation step, which consists of two independent actions that may go in parallel: mix the solid ingredients and beat the eggs. Thus, we use parallel decomposition to refine the specification (recipie). The result is Figure 9 with the associated relations as follows.

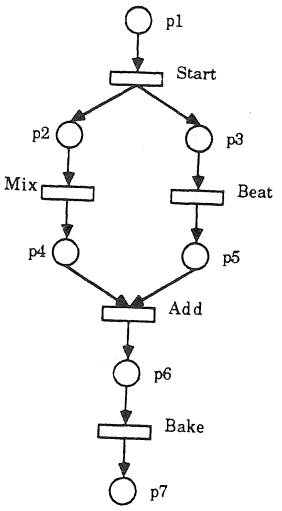


Figure 9: The recipie after parallel decomposition.

$$
\begin{aligned}
R(Start) &= \{[4(p_1,(egg,t_e)) \oplus (p_1,(flower,t_f)) \oplus (p_1,(butter,t_b)) \oplus (p_1,(sugar,t_s)),\\
&\quad (p_2,(flower,t_1)) \oplus (p_2,(butter,t_1)) \oplus (p_2,(sugar,t_1)) \oplus 4(p_3,(egg,t_1))]\}\\
&\quad where\, t_1 = Max(t_e,t_f,t_b,t_s)\\
R(Mix) &= \{[(p_2,(flower,t_f)) \oplus (p_2,(butter,t_b)) \oplus (p_2,(sugar,t_s)),(p_4,(mixture,t_2))]\}\\
&\quad where\, t_2 = Max(t_f,t_b,t_s) + 5\\
R(BtEgg) &= \{[4(p_3,(egg,t_e)),(p_5,(beateneggs,t_e+\alpha))]\}\ where\, \alpha \in \{5,10\}\\
R(Add) &= \{[(p_4,(mixture,t_m)) \oplus (p_5,(beateneggs,t_{be})),(p_6,(dough,Max(t_m,t_{be})))]\}\\
R(Bake) &= \{[(p_6,(dough,t_d)),(p_7,(cake,t_d+40))]\}\\
M_0 &= 4(p_1,(egg,0)) \oplus (p_1,(flower,0)) \oplus (p_1,(butter,0)) \oplus (p_1,(sugar,0))
\end{aligned}
$$

Finally, we can apply nondeterministic decomposition to the action "BeatEggs" and obtain the net of Figure 1. Here we have only made obvious the nondeterminism of the transition, obtaining an equivalent net (not a refinament).

6 Conclusions

In this paper we proposed a formal definition of equivalence and refinement for ER-nets, a kernel formalism adequated for modelling concurrent and real-time systems. In particular we have presented different schemata of transformation for ER-nets that may be seen as a controlled way of doing refinements. Sequential and parallel decomposition may become useful mechanisms for the designer during the developement of a system specification. Certainly, some other transformations may be defined based on the given notion of refinement or directly in terms of the proposed ones.

The notion of refinement and the topological transformations, may be extended to allow also the coarsening of an ER-net by means of a higher abstract view. It would be also possible to define analysis techniques (such as symbolic excecution), that can be carried out at different abstraction levels.

Acknowledgements

We would like to thank Professor Carlo Ghezzi, who introduced us to the subject. We are also grateful to Miguel Felder, who made interesting suggestions and helped with the bibliography. The first author wish to thank his friends Marcelo Fernandez, Marcelo Fiore, Gabriela Anastasi and Sergio Yovine for their comments and fruitful conversations about this work.

References

[Berthomieu 83] Berthomieu B., Menasche M., An Enumerative Approach for Analyzing Petri Nets, *Information Processing 83*, R. E. A. Mason (Ed.), Elservier Science Pub. B.V., North Holland, 1983.

[Degano 89] Degano P. Meseguer J., Montanari U., Axiomatazing Net Computations and Processes, *4th. Annual Symposium on Logic in Computer Science*, Asilomar, CA, U.S.A., June 1989.

[Felder 88] Felder M., Some Properties when Time is Introduced in Petri Nets, Graduation Thesis, ESLAI, December 1988.

[Feuerstein 88] Feuerstein E., Timed Petri Nets are Monoids, Graduation Thesis, ESLAI, December 1988.

[Genrich 86] H. J. Genrich, Predicate Transition Nets, in *Advances in Petri Nets 1986*, LNCS 254-255, Springer Verlag 1987.

[Ghezzi 89] Ghezzi C., Mandrioli D. Morasca S., Pezzè M., On Introducing Time in Petri Nets, *IEEE 5th. International Workshop on Software Specification and Design*, Pittsburgh, U.S.A., May 1989.

[Ghezzi 89b] GhezziC., Mandrioli D., Morasca S., Pezzè M., Symbolic Excecution of Concurrent Programs Using Petri Nets, *Computer Languages*, vol. 14, No. 4, 1989.

[Ghezzi 91] GhezziC., Mandrioli D., Morasca S., Pezzè M., A Unified High Level Petri Net Formalism for Time Critical Systems, *IEEE Transactions of Software Engineering*, March 1991.

[Jensen 86] F. Jensen, Coloured Petri Nets, in *Advances in Petri Nets 1986*, LNCS 254-255, Springer Verlag 1987.

[Harel 86] Harel D., A Visual Formalism for Complex Systems, *Science of Computer Programming*, vol. 8, 1986.

[Meseguer 88] Meseguer J., Montanari H., Petri Nets are Monoids. SRI-CLS-88-3, SRI Proyect No. 1243 and 4415, January 1988.

[Morasca 89] Morasca S., Pezzè M., A Net Based Model for Time-Dependent Systems, Internal report No. 89-005, Dipartamento di Elettronica, Politecnico di Milano, Milan, Italy, 1989.

[Peterson 77] Peterson J., Petri Nets, Computing surveys, Vol. 9, No. 3, September 1977.

[Reisig 82] Reisig W., *Petri Nets, an introduction*, ETACS Monographs on Theoretical Computer Science; Springer Verlag 1985.

Uma experiência na implementação de operadores da álgebra relacional no computador paralelo NCP I

Marta L. Queirós Mattoso
Claudio Luis de Amorim
Programa de Engenharia de Sistemas e Computação
COPPE/UFRJ

Este trabalho descreve a experiência obtida com a implementação de um protótipo de sistema de banco de dados paralelo, o PARBASE sobre a máquina paralela desenvolvida na COPPE, o NCP I, utilizando um ambiente de programação paralela. São levantadas as diversas características que exploram o paralelismo num sistema de banco de dados. A experiência da implementação do PARBASE é comentada através dos operadores da álgebra relacional de seleção e junção. Foram obtidas diversas medidas de desempenho onde verificou-se um comportamento uniforme no PARBASE em relação à variações no tamanho das relações envolvidas. Este trabalho também mostra o ganho no desempenho quase sempre linear em relação ao acréscimo de processadores à configuração do NCP I.

1 INTRODUÇÃO

O desempenho de aplicações de banco de dados em máquinas sequenciais vem se tornando um fator crítico para aplicações que envolvem consultas complexas e relações com grande volume de dados. O processamento de operações de banco de dados oferece várias oportunidades de paralelismo que apontam um ganho no tempo de resposta dos sistemas de banco de dados (SBD). A habilidade de se explorar um grande número de processadores conectados em paralelo é hoje uma área de pesquisa intensa. A contribuição do processamento distribuído para banco de dados está no acesso paralelo ao armazenamento das bases de dados e no processamento paralelo desses dados. A dificuldade de se sintonizar um sistema de banco de dados paralelo (SBDP), está intimamente ligada ao balanceamento de carga entre os processadores e à minimização do custo de coordenação e sincronização [Laks 90]. O panorama atual dos SBDPs tem muito a contribuir com experências práticas ou analíticas para as diversas fontes de paralelismo na gerência de bases de dados.

No sentido de se ganhar experiência na distribuição e sincronização de operações da álgebra relacional, foi implementado na máquina paralela desenvolvida na COPPE, o NCP I [Amor91], um protótipo de SBDP, o PARBASE, utilizando o ambiente de programação paralela STRAND [Fost90]. Realizaram-se várias medidas de desempenho no PARBASE onde verificou-se o ganho no desempenho sempre que o número de processadores envolvidos no processamento da consulta aumentava, além de ser observada a estabilidade do sistema em relação à variação do número de tuplas nas relações envolvidas.

Neste trabalho, são apresentadas na seção 2, as principais características e técnicas envolvidas no projeto de SBDPs atuais. A seção 3 apresenta o ambiente de desnvolvimento do protótipo PARBASE. As características, os componentes e a implementação do PARBASE são descritos na seção 4. Comentários sobre o desempenho e comportamento do PARBASE são realizados na seção 5 com o auxílio de gráficos de

tempo de resposta e de ganho em desempenho. Finalmente a seção 6 conclui o trabalho com a experiência obtida.

2 ESTAGIO ATUAL DOS SBDs PARALELOS

As pesquisas na área de paralelismo em sistemas de banco de dados surgem a partir da tecnologia de sistemas de banco de dados distribuídos e das máquinas de banco de dados. Embora essas duas áreas de conhecimento em banco de dados venham sendo exploradas há muito tempo, recentemente elas ganharam um novo impulso com a disponobilidade da tecnologia atual das arquiteturas paralelas, gerando os chamados sistemas de banco de dados paralelos (SBDP). Uma taxonomia para diversos protótipos/produtos encontrados na literatura de acordo com as diversas fontes de paralelismo mencionadas neste trabalho, pode ser encontrada em [Matt91]. Esta taxonomia visa identificar aspectos que são úteis para a comparação, descrição e desenvolvimento de SBDPs.

2.1 Categorias de SBDPs

Os SBDPs atuais caracterizam-se por seus estágios de desenvolvimento como protótipo ou produto e ainda o seu acoplamento ao hardware. Existem basicamente três categorias de concepção de SBDPs descritas a seguir:

i) máquina de bd

As máquinas paralelas de banco de dados são totalmente acopladas ao hardware, ou seja, é projetado um hardware com capacidade de processamento paralelo, específico para suportar as operações de um sistema de banco de dados. Os produtos NonStopSQL [Tand88] e DBC/12 [Alma89] são exemplos representativos dessa categoria. Como protótipos, podem ser citados os sistemas Bubba [Bora90] e Gamma [DeWi90] entre outros.

ii) servidor de bd

Os servidores de sistemas de banco de dados paralelos caracterizam-se por seu desacoplamento ao hardware. Em geral, esses servidores não utilizam um hardware específico para banco de dados, mas possuem um sistema operacional que é orientado para banco de dados. Muitas vezes, esse servidor está ligado a várias estações de trabalho que interagem com os usuários realizando um processamento local. Um exemplo típico desta categoria é o projeto do European Data Server (EDS) [Vald90].

iii) SBDP geral

Estes sistemas paralelos, são gerenciadores de banco de dados desenvolvidos para um supercomputador de uso geral, ou seja, embora o SBDP rode sobre a arquitetura de uma determinada máquina paralela, esta máquina é de uso geral e não específico para banco de dados. Como exemplo desta categoria está o protótipo XPRS de Stonebraker [Ston88].

2.2 Níveis de granularidade

Os SGBDs em geral, mais especificamente o relacional, oferecem várias oportunidades de paralelismo tanto na entrada/saída de muitos dados quanto no processamento de consultas. As pesquisas de SGBDs paralelos se concentram nas áreas de particionamento de dados - para aumentar o desempenho da entrada/saída - e no processamento de consultas. Este por sua vez pode ser dividido nos níveis de granularidade a seguir:

i) Entre consultas:

Caracteriza-se pela execução de consultas de diversos usuários concorrentemente. O objetivo da implementação paralela é de atender o maior número possível de consultas por segundo de forma concorrente.

ii) Dentro de consultas:

O objetivo é de atender o mais rápido possível ao processamento de consultas complexas e ad-hoc. Em geral, aplicações não convencionais (do tipo CAD, IA, Engenharia) caracterizam-se por possuirem poucas, longas e complexas transações exigindo assim um alto grau de paralelismo dentro da máquina para a própria transação ou consulta complexa. O paralelismo dentro da consulta está ligado à execução de operações de uma mesma consulta em paralelo.

iii) Dentro de operações:

Quando uma consulta é mapeada para as operações correspondentes que a resolvem, existem uma série de algoritmos incorporados ao SGBD que executam essas operações. Nesse nível de granularidade, procura-se paralelizar o algoritmo responsável pela resolução da operação, explorando ao máximo a disposição (particionamento) dos dados. No caso do modelo relacional, essas operções compõem a álgebra relacional, e o paralelismo dentro das operações da álgebra consiste, em geral na replicação da operação entre os processadores, e na excução da mesma operação para as partes da relação. Encontra-se na literatura um vasto material contendo algoritmos paralelos para os diversos operadores da álgebra relacional [Bitt83], [Vald84], alguns visando explorar ao máximo determinadas classes de arquiteturas como em [Rich87] e [Frie90], ou ainda explorando modelos de memória onde novos algoritmos são propostos [Chei88], [Schm89], [Murp89].

2.3 Modelos de memória

A organização da memória (principal e disco) têm influência decisiva no projeto do servidor de dados de um SBDP. Os algoritmos utilizados e principalmente a comunicação para entrada/saída costumam ser direcionados para o modelo de memória compartilhada ou para o modelo de memória distribuída (troca de mensagens). Stonebraker [Ston86] batizou esses dois modelos respectivamente de 'shared everything' (armazenamento compartilhado), já que tanto a memória principal quanto o armazenamento em disco são compartilhados por todos os processadores da arquitetura paralela e de 'shared nothing' (armazenamento não compartilhado), já que cada processador possui a sua memória local exclusiva e suas unidades de disco

também de acesso exclusivo. Existe ainda a denominação 'Shared Disk' [Bhid88b] para um modelo intermediário onde os processadores possuem memória local mas o espaço em disco é compartilhado.

Não existe um consenso em relação ao modelo mais eficiente. Alguns trabalhos analíticos [Bhid88a,b] e [Laks90] apontam a arquitetura com memória compartilhada como a solução que atinge o maior desempenho. Entretanto, vários projetos como Gamma [DeWi90], Bubba [Bora90], EDS[Vald90], entre outros apostam na expansibilidade da memória distribuída.

A seguir serão apresentadas algumas características dos modelos [Ozsu91], [Alma89], [Ston88], comparando-os do ponto de vista do projetista de um SGBD (já que para o usuário final o modelo da memória é transparente).

i) Arquitetura com memória compartilhada (Shared Everything)

A memória compartilhada é um ponto de estrangulamento em potencial. Se ocorre uma falha no esquema de interconexão da memória com os processadores, o sistema entra em pane. A arquitetura permite poucas extensões para o hardware possuindo um número limitado de processadores, uma vez que são dependentes da largura da banda ('bandwidth') da tecnologia do barramento ('bus') e da memória. Entretanto, na memória compartilhada, o particionamento dos dados é muito dinâmico pois os dados não estão distribuídos fisicamente permitindo um bom balanceamento de carga, já que num SGBD este fator está intimamente ligado à distribuição dos dados entre os processadores.

ii) Arquitetura com memória distribuída (Shared Nothing)

Permite expansibilidade para a arquitetura, o software e o hardware são projetados prevendo a extensão do número de processadores. Aumenta a disponibilidade dos dados através do isolamento de falhas e replicação. A programação do projetista do SGBD, de um modo geral, é mais complexa pois existe a necessidade de controle dos dados replicados, e obriga um controle adicional de concorrência para a cooperação de processos. Pode aumentar o desempenho aparente da entrada/saída dos dados através da realização de acessos paralelos às unidades de disco evitando o uso dos controladores de disco do processador dedicado à ES. Entretanto, o grau de paralelismo vai depender do particionamento dos dados entre os discos associados aos processadores.

2.4 Particionamento dos dados

O desempenho de uma operação está diretamente ligado com a fragmentação no armazenamento dos dados de uma base. Os algoritmos levam em consideração essa distribuição no sentido de se obter o melhor balanceamento de carga entre os nós. O balanceamento é ainda mais crítico nas arquiteturas do tipo 'shared nothing' uma vez que a alocação dos dados é em geral física, realizada no momento do carregamento da base. Neste caso, para grandes volumes de dados o ideal é se executar a operação onde o dado estiver, para evitar grande tráfego na troca de mensagens. Existem dois grandes tipos de particionamento de dados o agrupado ('clustering') e o desagrupado ('declustering') [Ozsu91]. No particionamento **agrupado**, cada relação fica totalmente "contida" em um mesmo nó. Essa técnica poderá minimizar o tempo total de execução se todas as relações envolvidas estiverem no mesmo nó. Entretanto, o particionamento **desagrupado** é o mais utilizado, onde uma relação é fragmentada horizontalmente através dos nós. Neste caso, o tempo de resposta

provavelmente será minimizado, porém o tempo total poderá aumentar devido ao custo de manter a relação fragmentada através de uma determinada função.

A fragmentação horizontal pode ser feita de diversas formas e a informação sobre o particionamento da relação é armazenado no catálogo. No caso de uma arquitetura com memória compartilhada ('shared everything') os diversos fragmentos são espalhados em arquivos separados através das unidades de disco.

Na arquitetura com memória distribuída ('shared nothing') os fragmentos são distribuídos entre as unidades de disco associadas aos processadores, ou seja, a técnica de particionamento leva em conta o número de processadores com unidades de disco associadas.

As principais técnicas de fragmentação horizontal são:

i) Circular

Cada tupla é associada sequencialmente a um fragmento e os fragmentos são percorridos circularmente ('round-robin').

ii) Faixa de valores

As tuplas são fragmentadas de acordo com o valor de um determinado atributo dentro de uma faixa de valores. Por exemplo, empregados com salário entre 1.000 e 5.000 e empregados com salário acima de 5.000.

iii) Hashing

As tuplas são fragmentadas de acordo com o valor de um determinado atributo aplicado numa função de hashing. Nesta estratégia uma função randômica é utilizada para relacionar a unidade de armazenamento (o fragmento) com a tupla da relação. Por exemplo, seja uma arquitetura do tipo 'shared nothing' com 4 processadores, cada um com suas unidades de disco. Uma estratégia seria utilizar uma função randômica que gerasse valores entre 1 e 4 e as tuplas seriam então fragmentadas nesses 4 grupos ('buckets') associados aos quatro processadores.

Além das técnicas de fragmentação da relação, outras decisões tem que ser tomadas ao longo do particionamento dos dados. Essas decisões envolvem a associação da relação aos processadores [Cope88], a utilização de memória cache e a utilização de índices.

3 O AMBIENTE DE DESENVOLVIMENTO DO PARBASE

O ambiente de desenvolvimento do PARBASE pode ser dividido em duas partes. A primeira é o ambiente computacional caracterizado pela máquina paralela da COPPE/UFRJ, o NCP I. A segunda consiste no ambiente de programação englobando o sistema operacional paralelo Helios [Heli88] e o ambiente de programação paralela STRAND [Fost90], [Arti90]. O sistema operacional Helios é uma versão paralela do Unix. A vantagem da utilização do Helios está na sua portabilidade, pois sendo genérico e

rodando em diversas máquinas, facilita a integração do SBDP com outros sistemas. STRAND é uma linguagem de programação com características avançadas tanto para a programação sequencial convencional quanto para a programação paralela.

3.1 Ambiente computacional

O computador paralelo NCP I [Amor91] é um protótipo pré-industrial projetado e implementado na COPPE/UFRJ. O NCP I é baseado numa arquitetura de memória distribuída com topologia básica hipercúbica. A comunicação entre os nós de processamento é feita através dos elos seriais de comunicação à uma taxa de 20 Mbits/s. A arquitetura de um nó é mostrada na Figura 1. Como pode ser visto, cada nó possui um transputer T800 (32 bits, 25 MHz com 4 Mbytes de memória e um microprocessador i860 (64 bits, 40 MHz) com 8 Mbytes de memória, esta sendo compartilhada com o transputer. Além disso, existem 16 Kbytes de memória vetorial para troca rápida de informação entre os dois processadores. A comunicação externa é feita através de um crossbar switch e de um barramento VME. O primeiro permite a interligação do nó com outros 2048 nós, utilizando comunicação serial. Este crossbar switch permite desacoplar a arquitetura da topologia hipercúbica quando necessário. A interface VME fornece uma comunicação paralela à base de dados podendo ser utilizada também para comunicação entre nós vizinhos. Neste barramento pode ser colocado, além de controladores de disco, interface para rede e interface para outros computadores hospedeiros. A capacidade do protótipo é de 1,28 GFlops, 800 Mips e 192 Mbytes de memória.

O modelo T8 do NCP I utilizado neste trabalho, corresponde a 8 nós interligados nessa topologia hipercúbica. Cada nó contém um transputer e 2 Mbytes de memória. O computador hospedeiro é um IBM PC-AT. A capacidade do T8 é de 16 MFlops, 80 Mips e 16 Mbytes de memória.

3.2 Ambiente de programação STRAND

i)STRAND como linguagem de programação convencional

STRAND é uma linguagem declarativa e que impõe recursividade em sua programação. Possui tipos de dados estruturados como listas, tuplas e conjuntos, o que facilitou o desenvolvimento do PARBASE. Entretanto, todas as suas variáveis são univaloradas, o que acrescenta alguma dificuldade de programação. STRAND é interpretada e foi projetada para a programação multi-linguagens, ou seja, possui mecanismos de integração com Fortran e C para o desenvolvimento de código local sequencial e reaproveitamento de código. A linguagem tem sabores de Prolog e Smalltalk, mas não chega a implementar os paradigmas da programação em lógica ou da orientação a objetos.

ii)STRAND como linguagem de programação paralela

Um programa STRAND caracteriza-se por um pool de processos se comunicando através de variáveis que são argumentos dos processos. Os processos são obtidos a partir da execução de procedimentos que são escritos como regras da forma:

$r(p_1,..,p_n)$:- <guarda de cláusulas> | <corpo>.

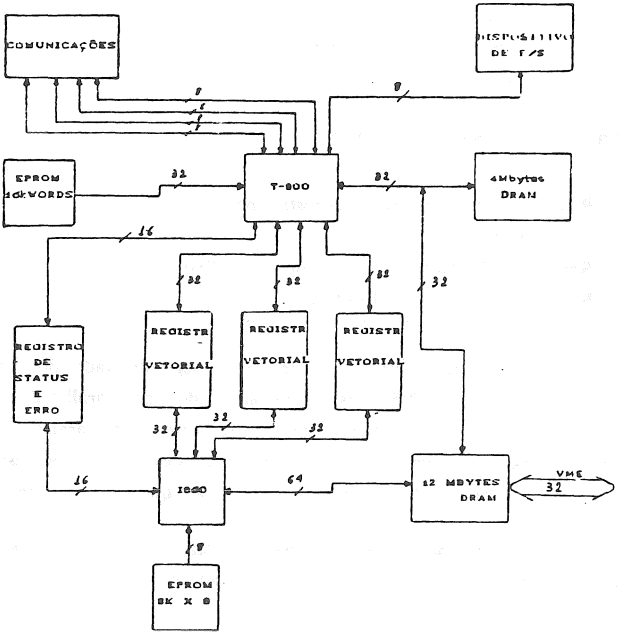


Figura 1 - Arquitetura do nó do NCP I

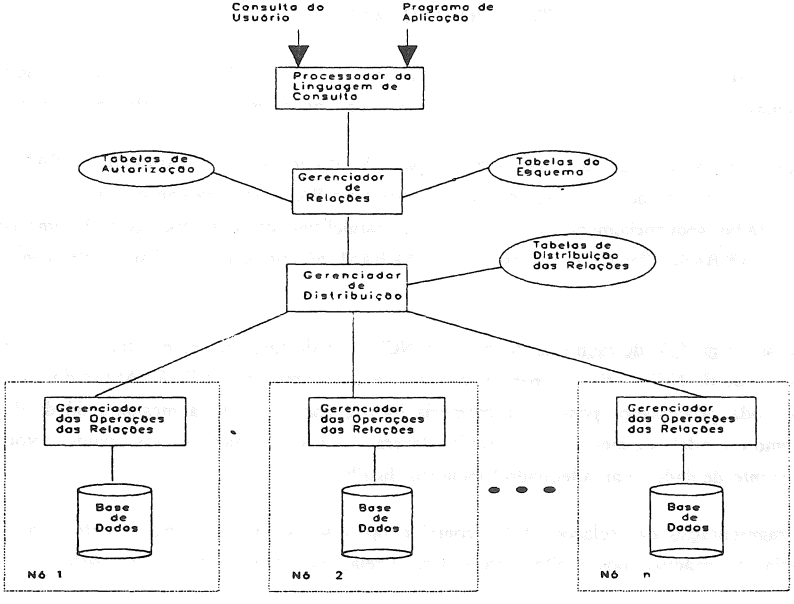


Figura 2 - Componentes do PARBASE

Onde r identifica uma regra de n parâmetros, ":-" é o símbolo de implica em, <guarda de cláusulas> são as cláusulas a serem verificadas para a execução do corpo, "|" delimita as cláusulas do corpo e o <corpo> é um conjunto de processos.

Um procedimento teste pode ser definido por:

```
teste(Numero, Resposta):- Numero >0 | Resposta:=positivo.
teste(0, Resposta):- Resposta:=zero.
teste(Numero, Resposta):- Numero <0 | Resposta:=negativo.
teste(Numero, Resposta):- otherwise | Resposta:=erro.
```

STRAND é altamente concorrente, aproveitando a disponibilidade dos processadores para executar (reduzir) os processos do pool. Não existem comandos do tipo 'parallel do', 'fork' ou 'join'. A princípio todos os processos podem ser executados concorrentemente e em paralelo, o que implica num sincronismo que é controlado pelas variáveis (os argumentos dos processos). Um processo que depende da instanciação de uma determinada variável, fica aguardando no pool até que ela receba um valor.

Outra característica interessante é a sua associação direta entre código e processadores. No caso de um SBDP, o código da seleção pode ser associado a n processadores diretamente para que a seleção seja executada em paralelo.

4 CARACTERÍSTICAS DO PROTÓTIPO PARBASE

PARBASE é um protótipo de sistema de gerência de bases de dados paralelo de uso geral que implementa alguns operadores da álgebra relacional e é independente da máquina paralela hospedeira.

O nível de granularidade do processamento paralelo de consultas mais explorado no PARBASE é o paralelismo dentro de operações. Entretanto, uma consulta no PARBASE não necessariamente terá suas operações executadas sequencialmente. É possível haver paralelismo entre as operações de uma consulta controlado pelo PARBASE. Por ser mono-usuário, o PARBASE não provê o paralelismo entre consultas de vários usuários.

Apesar do modelo de memória da máquina NCP I ser do tipo 'shared nothing' (armazenamento distribuído), a versão do NCP I utilizada para o desenvolvimento e execução do PARBASE é do tipo 'shared disk', ou seja, cada processador possui sua memória local e compartilham a mesma unidade de disco. Entretanto, como o modelo da memória da máquina deverá ser de armazenamento distribuído, optou-se por um particionamento de dados mais adequado à memória distribuída.

A fragmentação das relações é horizontal e optou-se por um desagrupamento total para as relações, ou seja, se a máquina possui oito processadores a relação é fragmentada em oito partes.

A política de distribuição de tuplas de uma relação entre os diversos fragmentos é a de faixa de valores. No momento da criação de uma relação o usuário deve especificar o atributo de distribuição e as

respectivas faixas de valores. Esta estratégia de distribuição assim como o hashing, tiram proveito da localidade conhecida para as faixas de valores e com isso atingem um ganho em desempenho.

4.1 A implementação do PARBASE

Os componentes do PARBASE estão representados na Figura 2. Essa implementação aproveita várias sugestões fornecidas por Steer [Stee89] para o desenvolvimento de um SGBD concorrente.

O usuário interage com o PARBASE através de comandos associados aos operadores da álgebra relacional, com a seguinte sintaxe:

```
select(Id-us, Atributos, Relação, Condição, Resultado)
join(Id-us, Relação1, Relação2, Condição, Resultado)
insert(Id-us, Relação, Tupla, Resultado)
delete(Id-us, Relação, Condição, Resultado)
update(Id-us, Relação, Condição, Atributos, Resultado)
create(Id-us, Relação, Chave, Especificação, Resultado)
drop(Id-us, Relação, Resultado)
```

No sentido de se facilitar o desenvolvimento do protótipo, não foi implementado um processador para analisar a linguagem de consulta. Assim a sintaxe elaborada simplifica as verificações léxicas e sintáticas no PARBASE. O usuário pode submeter um ou mais comandos através de uma lista ao PARBASE. O sistema se encarregará de prover o máximo de paralelismo entre os comandos da lista do usuário.

O Gerenciador de Relações, é responável por algumas verificações semânticas do tipo existência da relação envolvida, nível de autorização do usuário para determinadas operações, etc.

O Gerenciador de Distribuição é o responsável pela análise dos atributos envolvidos na operação e pela consequente decisão sobre quais processadores estarão envolvidos na execução da operação. O Gerenciador de Distribuição consulta as tabelas de distribuição e eventualmente impõe alguma sincronização na execução das sub-operações. Na identificação dos processadores e fragmentos para operações que envolvem condições, o Gerenciador de Distribuição verifica se somente um processador será ativado ou se todos executarão a operação, cada um com seu fragmento da relação. Ao término da operação, esse mesmo Gerente se encarrega de fazer a união da relação resultado. Quando somente um processador é envolvido na operação, os outros processadores executam as operações restantes da consulta do usuário.

No caso do operador de junção, o Gerente de Distribuição identifica se a junção pode ser executada totalmente distribuída ou se a junção será executada parcialmente distribuída. A execução da junção totalmente distribuída ocorre quando as duas relações envolvidas estão fragmentadas sobre seus respectivos atributos da condição de junção. Neste caso, as operações de junção são executadas em paralelo cada uma em um processador e é realizada a união dos resultados parciais no final. A execução da junção parcialmente distribuída ocorre em duas fases. Inicialmente o Gerenciador de Distribuição escolhe a relação de menor cardinalidade, junta todos os seus fragmentos para que em seguida a envie completa para todos os

processadores para que cada um execute a junção com a parcela da outra relação operando, associada ao seu nó.

5 ANÁLISE DO DESEMPENHO DO PARBASE

São apresentados aqui os resultados iniciais de uma avaliação do desempenho do protótipo do SBDP PARBASE na máquina paralela NCP I. Foram realizados uma série de testes com os operadores de seleção e junção onde foram analisados os tempos de resposta medidos para variações no número de tuplas das relações e no número de processadores utilizados na configuração da arquitetura.

As relações utilizadas para o benchmark são instâncias de uma base de dados real. Cada relação possui quatro atributos inteiros de 4-bytes e quatro atributos do tipo cadeia de caracteres totalizando 97 caracteres. Foram realizados testes sobre relações de 100, 1000 e 3000 tuplas. Cada relação possui dois atributos sem duplicatas e não estão relacionados entre si. Um desses atributos de valor único, foi utilizado para a chave de distribuição em todos os casos.

Todos os testes foram repetidos para configurações de 1, 4 e 8 processadores no NCP I. Os tempos foram medidos a partir do momento que o usuário submeteu a consulta ao PARBASE até a finalização da consulta. Foram incluídas neste tempo todas as passagens da consulta pelos diversos gerentes do PARBASE.

5.1 Seleção

O desempenho do PARBASE foi explorado para vários tipos de seleção sobre relações cujo tamanho também variava, baseado no benchmark do SBDP Gamma [DeWi88]. O objetivo dessas medidas foi o de analisar o comportamento do PARBASE à medida que o tamanho das relações aumentava. Em condições ideais, espera-se que o tempo de resposta aumente de acordo com uma função linear sobre o tamanho das relações operando, dado que as configurações da máquina são constantes.

Foram realizados testes sobre quatro tipos de consulta envolvendo a seleção. Em tres consultas variou-se o fator de seletividade com 0%, 1% e 10% para as relações resultado. Na quarta consulta o resultado da seleção envolvia somente uma tupla.

As tabelas 1 e 2 mostram os resultados obtidos para os diversos tipos de seleção sobre relações de tamanho 100, 1000 e 3000 tuplas para as configurações de 4 e 8 processadores ligados em hipercubo. Como as relações eram de uma base real foram realizadas pequenas adaptações para se atingir exatamente os fatores de seletividade.

	Número de tuplas da relação fonte		
Consulta	100	1000	3000
seletividade 0%	5,40	50,26	175,87
seletividade 1%	5,42	50,94	174,59
seletividade 10%	5,44	47,85	173,15
seleção 1 tupla	4,74	44,14	128,76

Tabela 1 - Consultas de seleção para 4 processadores
(todos os tempos em segundos)

	Número de tuplas da relação fonte		
Consulta	100	1000	3000
seletividade 0%	4,45	24,26	76,23
seletividade 1%	4,40	23,27	74,25
seletividade 10%	4,45	23,29	76,23
seleção 1 tupla	3,34	18,69	61,29

Tabela 2 - Consultas de seleção para 8 processadores
(todos os tempos em segundos)

Algumas conclusões podem ser obtidas das tabelas 1 e 2. Em primeiro lugar, conforme esperado, observa-se que o tempo de execução de cada consulta aumenta quase que proporcionalmente com o tamanho das relações operando para cada número de processadores testados, ou seja, 1, 4 e 8.

A única excessão ocorre para as seleções da relação de 100 tuplas quando aumentam para 1000 tuplas na configuração de 8 processadores. Nas configurações de 1 e 4, o tempo de resposta cresce em função do aumento de tuplas nas relações. Essa excessão é explicável pela má distribuição de carga que ocorreu entre a relação de cardinalidade 100 e a configuração de 8 processadores. Foi observado que não existe muito ganho em desempenho ao se utilizar muitos processadores para uma relação com poucas tuplas. Em casos assim, o preço que se paga para manter os diversos fragmentos e a comunicação entre os processadores não se traduz em custo/benefício. Esta é uma situação em que o desagrupamento total não funciona bem.

Como segunda conclusão pode ser observado o ganho em desempenho sempre que se aumenta o número de processadores e consequentemente o número de fragmentos que compõem a relação.

Cabe ainda observar que os melhores tempos de resposta estão sempre para a seleção de uma tupla, já que esta é a única consulta em que o Gerente de Distribuição do PARBASE tira proveito da distribuição da relação e não necessita percorrer toda a relação, somente o fragmento envolvido.

A Figura 3 apresenta o tempo médio de resposta para consultas com fator de seletividade 0%, 1%, 10% e seleção de 1 tupla, para a relação de 1000 tuplas em função do número de processadores utilizados. A Figura 4 apresenta a curva de ganho relativo ao tempo de processamento correspondente à Figura 3. Pode ser observado que o ganho é quase linear para os fatores de seletividade.

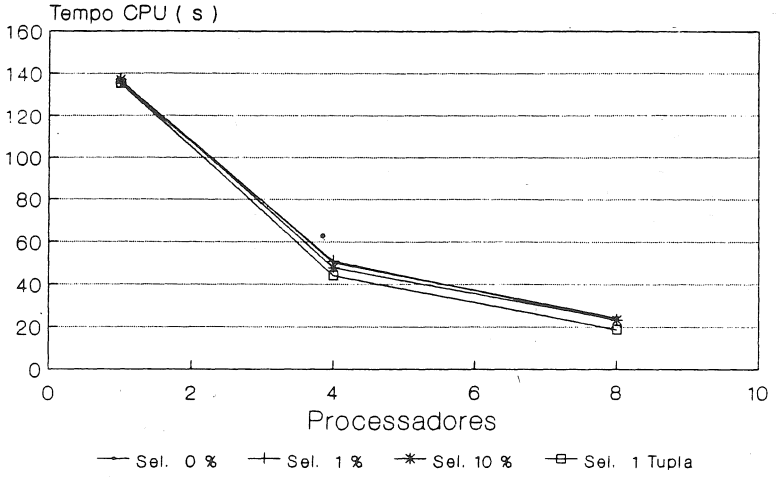


Figura 3 - Tempos de resposta base de 1000 tuplas.

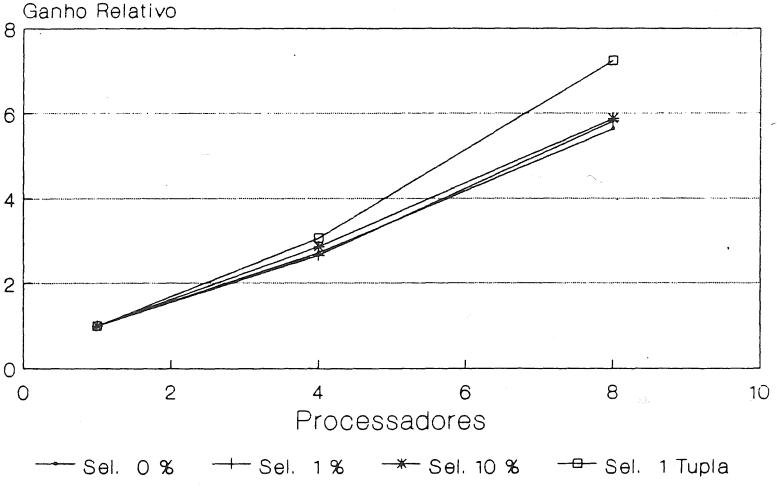


Figura 4 - Ganhos para a base de 1000 tuplas

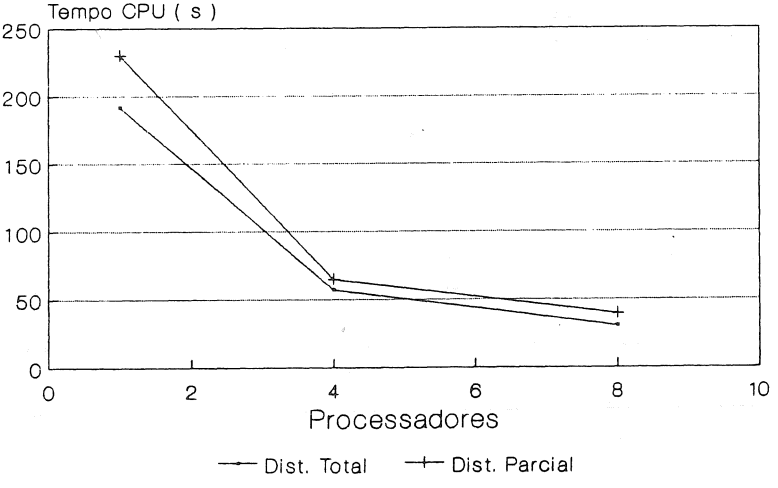


Figura 5 - Tempos de resposta junção de 100x1000 tuplas

5.2 Junção

Foram realizados diversos testes sobre o operador de junção do PARBASE e são apresentados aqui os resultados para junções ocorrendo entre tabelas de 100x1000 tuplas. O objetivo das medidas das consultas envolvendo o operador de junção foi o de analisar o desempenho do PARBASE à proporção que o número de processadores e fragmentos da relação aumentava.

Dois tipos de junção foram testados. No primeiro tipo, a junção opera sobre os atributos de distribuição das duas relações operando. Neste caso o PARBASE executa a operação de junção totalmente distribuída. No segundo tipo de teste, a junção opera sobre atributos não chave de distribuição das relações operando. Neste caso, o PARBASE executa a operação de junção em duas fases, sendo que a primeira é para a redistribuição da relação de menor cardinalidade.

Conforme era esperado, pode ser observado a melhora do desempenho através das curvas de tempo de resposta (Figura 5) e do ganho relativo. Estas curvas também mostram um tempo de resposta mais rápido para a junção totalmente distribuída. Este ganho só não é mais significativo devido ao fato da relação de menor cardinalidade conter apenas 100 tuplas.

5.3 Considerações sobre a implementação

O protótipo PARBASE foi implementado no sentido de se explorar técnicas de paralelismo para banco de dados e analisar o comportamento de um SBDP no computador paralelo NCP I. Para atingir esses objetivos foi realizada uma implementação do tipo prototipagem rápida (usufruindo de STRAND). Nessa implementação não houve preocupação com o desempenho em comparação com outros sistemas. Buscou-se a elaboração de um sistema que provê a funcionalidade mínima de um SGBD e que serve como uma plataforma onde diversas técnicas de paralelismo podem ser experimentadas, contribuindo para a construção de um SBDP que atenda necessidades reais.

Embora a implementação tenha atendido às expectativas de aceleração e estabilidade no seu comportamento, o PARBASE apresenta alguns pontos de estrangulamento, já conhecidos na fase de projeto e evidenciados nos testes realizados. Dentre esses pontos críticos, podemos citar:

AMBIENTE DE PROGRAMAÇÃO - Se por um lado a linguagem STRAND apresenta características que favorecem a prototipagem rápida, pois evita a programação de rotinas de comunicação, por outro lado, a comunicação entre os processadores pode não estar sendo realizada da maneira mais eficiente possível. Além disso a linguagem é interpretada e não foi explorada a sua facilidade de comunicação com outras linguagens. Na próxima versão do PARBASE pretende-se escrever todas as rotinas do Gerenciador de Operações sobre as relações em C, o que deverá diminuir o tempo de acesso aos dados.

ARMAZENAMENTO SECUNDÁRIO - O fato do NCP I estar utilizando na atual versão o disco rígido de um PC para o armazenamento secundário, faz com que a comunicação da entrada e saída seja um ponto de estrangulamento, pois todos os processadores competem para obter o mesmo recurso. Somado a isso está a ausência de métodos de acesso próprios ao PARBASE e de um esquema de bufferização.

ÍNDICES - A ausência de índices no processamento de consultas é um fator limitante no desempenho do PARBASE para diversas consultas. Segundo DeWitt [DeWi 88] a utilização de índices no SBDP Gamma fez com que o tempo de resposta fosse reduzido em aproximadamente 50% sendo observado em alguns casos reduções de até 25%. Cabe salientar que índices locais podem ser facilmente implementados no PARBASE.

ANALISADOR DA LINGUAGEM DE CONSULTA - Optou-se por uma forma de interação com o PARBASE intermediária entre uma linguagem de consulta tipo SQL e um código de máquina interpretável pelo Gerente de Relações. Assim, evitou-se por um lado o desenvolvimento de um analisador léxico e sintático, mas por outro lado algumas análises e transformações semânticas do código foram repetidas ao longo de diversas fase do PARBASE.

6 CONCLUSõES

Foram levantadas na seção 2 diversas fontes de paralelismo na gerência de bases de dados. A proposta do PARBASE difere basicamente das outras por dois fatores. O primeiro é a sua independência da máquina paralela hospedeira e do sistema operacional. O segundo fator é a obtenção das medidas de desempenho na plataforma real onde se deseja desenvolver o SBDP produto.

Optou-se pelo desenvolvimento de um protótipo simples com as características básicas de paralelismo, sobre o qual várias técnicas alternativas podem ser avaliadas e exploradas. Os gerenciadores do PARBASE dão tratamento uniforme para todas as operações implementadas facilitando a modificação e extensão do sistema. Várias alterações foram propostas no item 5.3 no sentido de contribuir para o desempenho do PARBASE em sua próxima versão.

Foram obtidas diversas medidas de desempenho e reportadas aqui algumas aferições para as operações de seleção e junção, já que representam respectivamente operações totalmente distribuídas e diretamente paralelizáveis e operações parcialmente distribuídas. Os resultados foram de acordo com o esperado, ou seja, verificou-se um crescimento uniforme no sistema a partir do tamanho das relações e um ganho no desempenho quase sempre linear em relação ao acréscimo dos processadores nos casos analisados.

Finalmente, o fato de se possuir um protótipo que não foi desenvolvido especificamente para um determinado sistema operacional ou máquina paralela, permite que: i) novas medidas de desempenho possam ser realizadas e comparadas em outras máquinas, ii) haja mais independência dos dados, e iii) a construção e migração de aplicações para o SBDP sejam facilitadas.

7 REFERÊNCIAS

[Alma89] Almasi,G.S., Gottlieb,A. "Highly Parallel Computing" Benjamin Cummings, 1989.

[Amor91] Amorim,C.L. e equipe "O computador paralelo NCP I" Relatório interno, COPPE-Sistemas/UFRJ, março 1991.

[Arti90] Artificial Intelligence Limited "STRAND^{88} User Manual" Buckingham Release, June 1990.

[Bhid88a] Bhide,A., Stonebraker, M. "A Performance Comparison of Two Architectures For Fast Transaction Processing" Proceedings of the 4th International Conference on Data Engineering, IEEE, Los Angeles, Feb 1988.

[Bhid88b] Bhide,A. "An Analysis of Three Transaction Processing Architectures" Proceedings of the 14th International Conference on Very Large Data Bases, pp. 339-350, Los Angeles, 1988.

[Bitt83] Bitton,D. Boral,H. DeWitt,D. Wilkinson,K. "Parallel Algorithms for the Execution of Relational Database Operations" ACM Transactions on Database Systems, v.8(3), pp. 325-353, Sept 1983.

[Bora90] Boral,H. et al. "Prototyping Bubba, A Highly Parallel Database System" IEEE Transactions on Knowledge an Data Engineering, v.2(1), pp. 4-24, March 1990.

[Chei88] Cheiney,J-P. Maindreville,C. "A Parallel Transitive Closure Algorithm Using a Hash-Based Clustering" INRIA Research Repport 950, Dec 1988.

[Cope88] Copeland,G. Alexander,W., Boughter,E., Keller,T. "Data Placemente in Bubba" Proceedings ACM SIGMOD International Conference on Management of Data, pp.99-108, Chicago, June 1988.

[DeWi88] DeWitt,D.J. Ghandeharizadeh,S. Schneider,D. "A Performance Analysis od the Gamma Database Machine" Proceedings ACM SIGMOD International Conference on Management of Data, pp.350-360, Chicago, June 1988.

[DeWi90] DeWitt,D. et al. "The GAMMA Database Machine Project" IEEE Transactions on Knowledge an Data Engineering, v.2(1), pp. 44-62, March 1990.

[Heli88] Helios Group at Perihelion Software Limited "The Helios Operating System" Distributed Software Ltd, Version 1, Release 1.0, August 1988.

[Fost90] Foster I., Taylor S. "Strand: New Concepts in Parallel Programming", Prentice Hall, 1990.

[Frie90] Frieder,O. "Multiprocessor Algorithms for Relational-Database Operators on Hypercube Systems" IEEE Computer, v.23(11), pp. 13-29, Nov 1990.

[Laks90] Lakshmi,M.S. Yu,P.S. "Effectiveness of parallel processing in database systems", Computer Systems Science and Engineering, v.5(2) pp.73-80, April 1990.

[Matt91] Mattoso, M.L.Q."Paralelismo e Banco de Dados: Um Levantamento" Relatório Técnico da COPPE/UFRJ, a ser publicado.

[Murp89] Murphy, M.C. Rotem,D. "Effective Resource Utilization for Multiprocessor Join Execution" Proceedings of the 15th International Conference on Very Large Data Bases, pp. 67-75, Amsterdam, August 1989.

[Rich87] Richardson,P.J. Lu,H. Mikkilineni,K. "Design and Evaluation od Parallel Pipelined Join Algorithms" Proceedings ACM SIGMOD International Conference on Management of Data, pp.399-409, San Francisco, May 1987.

[Schn89] Schneider,D. DeWitt,D.J. "A Performance Analysis of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment" Proceedings ACM SIGMOD International Conference on Management of Data, pp.110-121, Portland, June 1989.

[Stee89] SteerValduriez,P. "Distributed and Parallel Database Systems", Apresentação de Tutorial, 5. Simpósio Brasileiro de Banco de Dados, Rio de Janeiro, Abril 1990.

[Ston86] Stonebraker,M. "The Case for Shared Nothing" IEEE Database Engineering, v.9(1), March 1986.

[Ston88] Stonebraker,M. "The Design of XPRS" Proceedings of the 14th International Conference on Very Large Data Bases, pp. 339-350, Los Angeles, 1988.

[Tand88] Tandem Performance Group "A Benchmark of NonStop SQL on the Debit Credit Transaction" Proceedings ACM SIGMOD International Conference on Management of Data, pp.337-341, Chicago, June 1988.

[Vald84] Valduriez,P. Boral,H. "Join and Semijoin Algorithms for a Multiprocessor Database Machine" ACM trans. on Database Systems, v.9(1), March 1984

[Vald90] Valduriez,P. "Query Processing in the EDS Parallel Database System", Apresentação de Tutorial, 5. Simpósio Brasileiro de Banco de Dados, pp. 2-14, Rio de Janeiro, Abril 1990.

[Ozsu91] Ozsu,M. Valduriez,P. "Principles of Distributed Systems", Prentice-Hall, 1991.

Uma Representação Gráfica para o Modelo de Objetos do GEOTABA

Luiz Carlos Montez Monte

COPPE - SISTEMAS / UFRJ
Caixa Postal 68511
CEP: 21.945 - Rio de Janeiro - RJ
fones: (021)590-2552, (021)590-2652, (021)290-4540
fax: (021)290-6626
e-mail: COS10002@ufrj.bitnet

Resumo - **O Modelo de Objetos do GEOTABA está sendo definido de modo a poder ser implementado diretamente pelo sistema de gerência de objetos GEOTABA, apesar de possuir um alto nível que permitirá seu uso na modelagem de informação e sistemas. Os Diagramas de Objetos são uma notação gráfica usável na modelagem conceitual de bases de objetos do GEOTABA, representando visualmente os conceitos contidos no seu Modelo de Objetos. Ao mesmo tempo que os Diagramas de Objetos sejam adequados à representação de modelagens semântica da informação, eles devem ser mapeados trivialmente no Modelo de Objetos do GEOTABA. Este trabalho apresenta os Diagramas de Objetos, mostrando, através de um exemplo, como a semântica da informação contida em uma modelagem tradicional, utilizando um Diagrama de Entidades-Relacionamentos comum, pode ser representada pelos Diagramas de Objetos. É mostrado também como as noções introduzidas neste exemplo são capazes de modelar propriedades não representáveis pelos Diagramas de Entidades-Relacionamentos convencionais. Extensões aos Diagramas de Objetos estão previstas, de modo a incorporá-los a uma linguagem visual de definição e manipulação de objetos para o GEOTABA. Os Diagramas de Objetos têm sido empregados na especificação do metaesquema do GEOTABA e como ferramenta para ajudar a compreensão e o aperfeiçoamento do Modelo de Objetos do GEOTABA.**

1. Introdução

O sistema de gerência de objetos GEOTABA, em desenvolvimento no Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ, pode ser classificado como um banco de dados orientado a objetos comportamentalmente. A informação por ele gerenciada é organizada de acordo com um Modelo de Objetos [MONTE 90] que engloba regras de definição de estruturas e de procedimentos de manipulação da informação. O Modelo de Objetos do GEOTABA, conforme proposto, é uma ferramenta de alto nível semântico, capaz de ser usada diretamente na modelagem de sistemas e da informação por eles manipulada. Apesar disso, este modelo é implementado diretamente pelo sistema de gerência de objetos GEOTABA. Este atendimento simultâneo ao homem e à máquina deve privilegiar os fatores humanos, correspondendo à tendência atual.

A utilização intensiva da capacidade gráfica dos computadores tem sido carta fundamental para o aumento da aceitação dos sistemas pelos indivíduos. A criação de uma representação gráfica para o Modelo de Objetos é uma providência que visa incrementar a usabilidade deste modelo.

Objetivo - Este trabalho apresenta a notação DO - Diagrama de Objetos -, representação gráfica do Modelo de Objetos do sistema de gerência de objetos GEOTABA. Esta representação gráfica é usável na modelagem conceitual de bases de objetos do GEOTABA, seja esta realizada manualmente ou com auxílio de computador.

A função principal do DO pode ser comparada à função da representação gráfica DER - Diagramas de Entidades-Relacionamentos. Ambas pretendem poder representar modelagens semânticas de informação. Todavia, sem qualquer prejuízo quanto à capacidade de representar estruturas e restrições, sem impor mecanismos extravagantes que dificultem a modelagem ou a torne artificial demais, o DO tem que se manter diretamente mapeável no Modelo de Objetos do GEOTABA. Assim, um diagrama deve corresponder exatamente a um esquema de classes e objetos implementável no GEOTABA.

A representação gráfica proposta deve ser adequada a utilização a mão livre, não contendo símbolos difíceis de desenhar. A possibilidade de se ter ferramentas de software para auxílio a confecção desses diagramas permite encarar essa representação gráfica como uma verdadeira linguagem visual. Para tal, os DOs devem poder representar totalmente o Modelo de Objetos, não apenas o nível conceitual dele. Para isto ser possível, os diagramas devem ser estendidos com mecanismos para representação de outros níveis de abstração do modelo: lógico, externo e apresentação. A possibilidade de se representar métodos e de se executar interativamente mensagens aos objetos, completarão a linguagem visual.

Escopo do Trabalho - Este relatório se concentra na descrição das características dos Diagramas de Objetos comparáveis com as características dos Diagramas de Entidades-Relacionamentos tradicionais. Outras possibilidades dos Diagramas de Objetos, capazes de representar outros tipos de restrições de integridade e características do Modelo de Objetos não capturáveis pelos Diagramas de Entidades-Relacionamentos convencionais, não são discutidas aqui. O trabalho busca mostrar como toda a informação modelável através de um DER pode ser representada por um DO.

Embora os Diagramas de Objetos possam colaborar para uma melhor compreensão do Modelo de Objetos do GEOTABA, este não é descrito neste trabalho. Apenas os conceitos relevantes do Modelo de Objetos, referentes a comparação entre os dois tipos de diagramas, são aqui comentados. Para um pouco mais de informação sobre o Modelo de Objetos, é possível referir-se a [Monte 90].

Trabalhos Correlatos - Notações gráficas para representar modelagens orientadas a objetos têm sido propostas e discutidas, ainda que escassamente [Booch 91, Coad 90, Henderson 90, Alabiso 88]. Modelos de dados semânticos produziram várias notações gráficas para modelagem conceitual, por exemplo, GSM [Hull 87], DER [Chen 76], EER

[Teorey 86], FDM [Shipman 81, Dayal 84]. O aproveitamento de qualquer destas notações é impedido pelas características específicas do Modelo de Objetos do GEOTABA, tais como: distinção entre classes e coleções (incomum nos modelos semânticos); associações constantes e variáveis; definição de relacionamentos (inexistentes nas notações orientadas a objetos) através de mapeamentos; e herança e sobreposição de propriedades, inclusive de mapeamentos.

Organização do Artigo - Os Diagramas de Objetos são ferramentas para modelar conceitualmente informação de acordo com características do Modelo de Objetos do GEOTABA. Portanto, certas peculiaridades do Modelo de Objetos, referentes ao nível conceitual de abstração, impõem requisitos sobre a representação gráfica, isto é, sobre os diagramas. Outros requisitos atendidos pelos Diagramas de Objetos se referem a características desejáveis de manipulação dos diagramas pelo usuário. A seção 2 enumera características relevantes do Modelo de Objetos e características de utilização pelo usuário e os requisitos correspondentes.

A seção 3 apresenta as noções envolvidas nos Diagramas de Objetos através da modelagem conceitual da informação de um exemplo típico utilizado na literatura para estabelecer as propriedades dos Diagramas de Entidades-Relacionamentos. Finalizando, na seção 4, são apresentados um sumário, perspectivas futuras e conclusões sobre o apresentado.

2. O Modelo de Objetos do GEOTABA

O Modelo de Objetos do GEOTABA é uma ferramenta conceitual equivalente a uma linguagem com semântica definida, porém sem uma sintaxe particular. Para o mesmo modelo de objetos, diferentes sintaxes podem ser estabelecidas. A notação gráfica proposta pode ser vista como uma possível sintaxe para simbolizar alguns conceitos do Modelo de Objetos.

Os Diagramas de Objetos aqui apresentados são uma sublinguagem dessa linguagem gráfica. Apenas a parte declarativa de definição de objetos é contemplada neles. A possibilidade dos diagramas poderem representar esta parte do Modelo de Objetos impôs requisitos específicos ao projeto destes diagramas. A seguir estão enumerados alguns dos requisitos para a representação gráfica relativos ao Modelo de Objetos.

Classes e Coleções - O Modelo de Objetos separa o conceito de classe de objetos do conceito de coleções de objetos. Classes reúnem apenas descrições de propriedades comuns a determinados objetos. Uma coleção é um depósito de objetos. As consultas, em geral, se referem às coleções.

O diagrama deve permitir a representação da definição de classes e, coletivamente, de esquemas de classes. O diagrama, além disso, deve permitir a definição de objetos globais persistentes, em geral coleções que formam bases de objetos, e, coletivamente, de esquemas de objetos. O diagrama deve poder ser visto em abstrações diferentes. Pode-se usá-lo tanto para representar um esquema de classes, como para representar um esquema de objetos, ou ainda para representar simultaneamente os dois esquemas.

Constantes e Variáveis - O Modelo de Objetos utiliza associações de símbolos com os objetos para que os usuários possam manipular os objetos através de nomes com significado adequado. O diagrama deve poder representar símbolos como nomes ou expressões em português.

Uma associação (símbolo, objeto) que, uma vez criada, não é mais mudada, isto é, o objeto é sempre o mesmo, é chamada de uma associação constante. O objeto envolvido em uma associação constante pode ser modificado internamente, contanto que continue sendo sempre o mesmo objeto (o mesmo identificador intrínseco de objetos). Isto é o que ocorre comumente com as coleções que formam uma base de objetos. Cada coleção tem um nome sempre associado a ela, mas a coleção em si pode variar. Um objeto que não varia seria um objeto constante.

Uma associação cujo símbolo pode, em instantes diferentes, estar associado a objetos diferentes é chamada de associação variável ou simplesmente de variável. Os diagramas devem permitir a representação distinta de associações constantes e de variáveis.

Relacionamentos - As associações internas de um objeto, denominadas variáveis de instância, implementam relacionamentos entre objetos. No nível conceitual, estes relacionamentos são classificados em atributos do objeto, componentes e mapeamentos. Os Diagramas de Objetos devem representar distintamente os diversos tipos de relacionamento. Todavia, a semelhança entre os diversos tipos de relacionamento deve ser notável nas suas representações gráficas.

A principal semelhança é que um relacionamento binário define um mapeamento de um objeto em outros. Este relacionamento pode definir um outro mapeamento, inverso do anterior. Neste caso, os dois mapeamentos formam um único relacionamento e o vínculo entre eles, nos diagramas, deve ser evidente. Descrevendo um relacionamento entre objetos, em cada objeto envolvido, através de atributos representando os seus mapeamentos, permite se ter uma visão orientada a objetos dos relacionamentos.

Relacionamentos não binários e relacionamentos com atributos não têm suporte direto no Modelo de Objetos. Eles são modelados como objetos quaisquer. Nos diagramas, a representação destes relacionamentos deverá se aproximar mais das representações de objetos comuns do que de relacionamentos binários.

CEDINFOR

Cardinalidade dos Mapeamentos - Também de acordo com o modelo, os mapeamentos têm nome e cardinalidade mínima e máxima. Assim, um mapeamento pode ser opcional ou obrigatório e ser univalorado ou multivalorado. Estas diferenças devem ser explícitas nos diagramas. Um mapeamento opcional significa que o objeto pode ser associado ao objeto nulo. Um mapeamento multivalorado, de acordo com o Modelo de Objetos, é o relacionamento do objeto com uma coleção de objetos e assim ele deve ser representado no diagrama.

Domínios Predefinidos - Por último, os mapeamentos podem ser feitos para objetos definidos na modelagem ou para objetos de domínios predefinidos do sistema. Os diagramas devem ter formas diferentes para representar mapeamentos em domínios triviais predefinidos pelo sistema, mapeamentos em objetos definidos no próprio diagrama e mapeamentos em objetos que não sejam do sistema, mas já tenham sido definidos. Este último tipo de representação de mapeamentos é usável no desenvolvimento incremental dos esquemas.

Serviços - Além dos atributos, componentes e mapeamentos, um objeto pode definir serviços, ou seja, processamento. Estes devem poder ser representados nos diagramas.

Classes e Herança - No Modelo de Objetos, a classificação relaciona um objeto à sua classe. Deve ser simples a identificação das classes dos objetos representados em um diagrama. A herança de propriedades de uma superclasse pelas suas subclasses deve poder ser representada nos diagramas. Do mesmo modo deve ser possível se representar a redefinição de propriedades em uma subclasse. As pseudoclasses são classes que não têm instâncias, são usadas apenas para serem superclasses de outras classes. As pseudoclasses devem ser representadas distintamente das demais classes.

Chaves - O Modelo de Objetos do GEOTABA suporta a definição de restrições de integridade sobre coleções de objetos. Por exemplo, um determinado atributo dos elementos de uma dada coleção pode ser uma chave para esta coleção, ou seja, não aceitar valores nulos, nem ter dois elementos na coleção com valores iguais nesse atributo. Os diagramas devem prover mecanismos de definição de quais atributos são chaves em uma coleção.

Nomes Implícitos - O modelo é bastante flexível quanto a necessidade de se dar nomes às entidades modeladas. Assim, pode-se ter classes sem nomes e o mesmo nome poder identificar entidades de tipos diferentes: classes, coleções, atributos, mensagens, etc. A representação gráfica deve simplificar ao máximo a tarefa do seu usuário de escolher nomes para as coisas. Isto pode ser alcançado definindo-se, na representação, regras para a definição de nomes , isto é, na ausência de uma especificação de nome feita pelo usuário. Entretanto, o usuário poderá definir nomes adequados quando bem desejar.

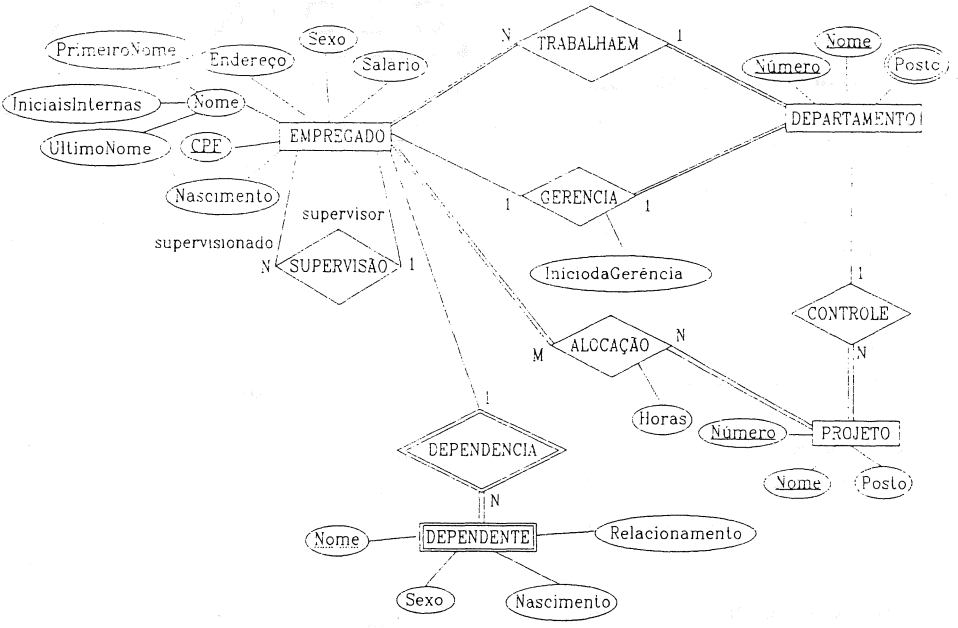


Figura 1: Diagrama Entidade-Relacionamento para a Companhia

Níveis de Visualização - Outros requisitos podem ser colocados independentemente do Modelo de Objetos. Por exemplo, a representação gráfica deve permitir múltiplos níveis de abstração. Uma mesma modelagem realizada com esses diagramas pode ter várias maneiras de ser apresentada, dependendo do tipo de informação que se deseja visualizar.

Os diagramas não devem obrigar o usuário a fazer declarações e definições que possam ser inferidas. Desse modo, a representação gráfica deve ser lacônica, sem prejuízo para a sua expressividade.

3. Um Exemplo de Modelagem

Este trabalho enfoca soluções contidas nos Diagramas de Objetos para que possam ser utilizados na modelagem conceitual da estruturas dos objetos de uma base. Sob esse aspecto, os Diagramas de Objetos podem ser comparados aos Diagramas de Entidades-Relacionamentos. Todavia, os Diagramas de Objetos devem representar o mais diretamente possível os esquemas implementáveis pelo Modelo de Objetos do sistema de gerência de objetos GEOTABA.

As características dos Diagramas de Objetos serão mostradas através de um exemplo. Visando a comparação com os Diagramas de Entidades-Relacionamentos, o exemplo escolhido é um já utilizado para exemplificar estes diagramas [Elmasri 89]. O

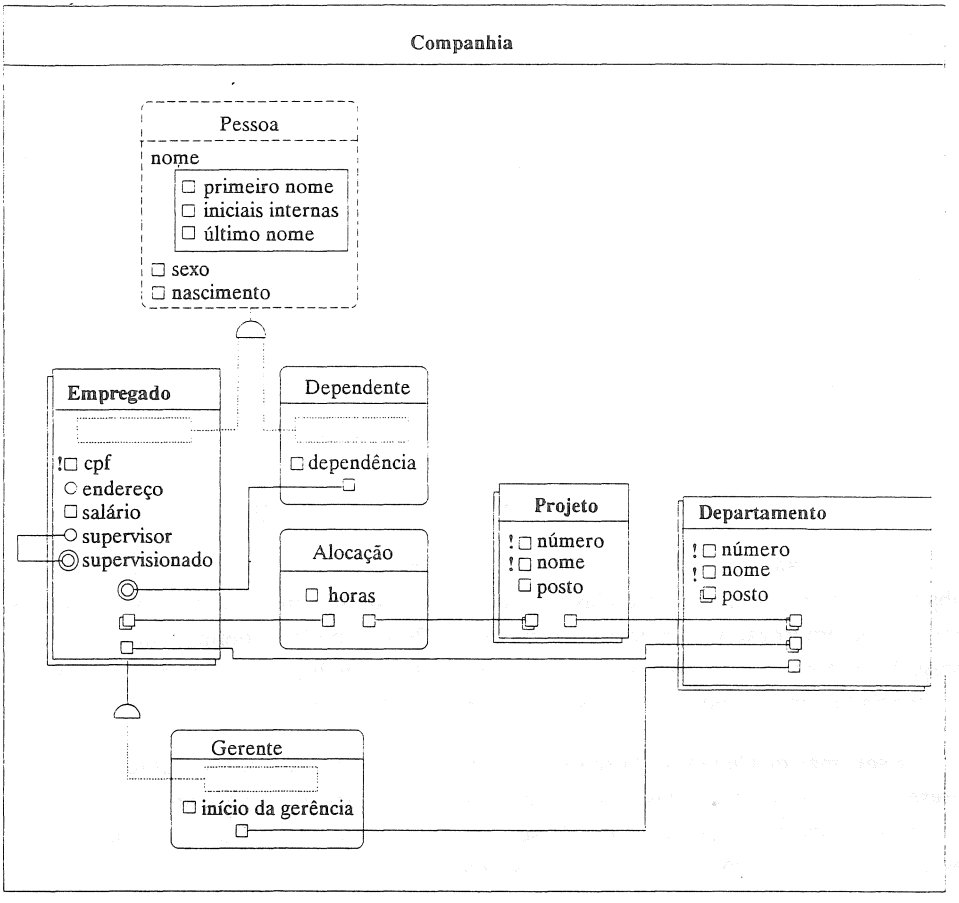


Figura 2: Diagrama de Objetos para a Companhia

exemplo descreve uma base de dados COMPANHIA, através do Diagrama Entidade-Relacionamento da Figura 1, que armazena informação sobre os empregados, departamentos e projetos de uma companhia. Os seguintes requisitos são representados no Diagrama Entidade-Relacionamento:

A Figura 2 representa, com um Diagrama de Objetos, o esquema de objetos e classes para uma base de objetos Companhia, de acordo com o Modelo de Objetos do GEOTABA. Através desse exemplo, serão mostradas, a seguir, as soluções empregadas pelos Diagramas de Objetos para atender aos requisitos estabelecidos anteriormente.

Notações de Classes e Coleções - O Diagrama de Objetos da Figura 2 representa simultaneamente os esquemas de classes e de objetos. As entidades Empregado, Projeto e Departamento estão representadas como coleções globais de objetos, contrastando com

Pessoa, Dependente, Alocação e Gerente que são classes cujos objetos não estão

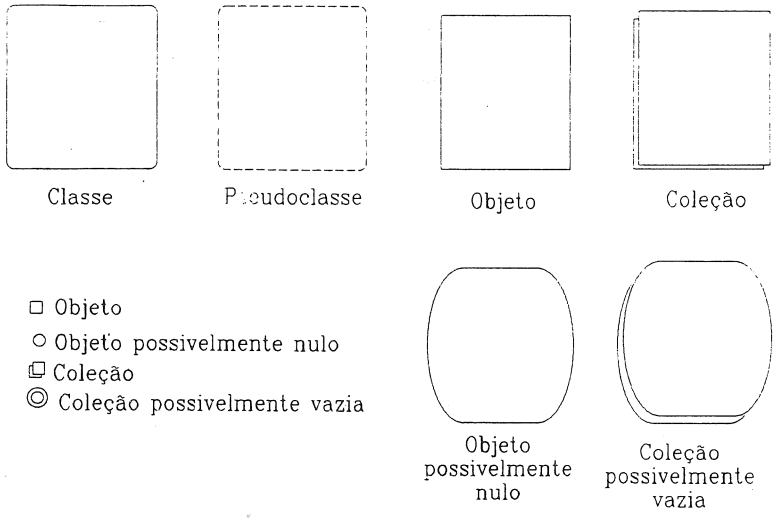


Figura 3: Ícones e Caixas para Classes e Objetos

reunidos em uma única coleção. A Figura 3 mostra as notações diferentes de classes e objetos. Com a distinção entre classes e coleções pode-se representar as propriedades comuns de empregados e dependentes na classe Pessoa, pode-se definir que certos empregados são gerentes e pode-se dispensar uma coleção global de dependentes. Assim, a existência de um dependente depende da sua associação a um empregado.

Esquemas de Classes e Coleções - A representação gráfica pode ser usada para representar o esquema de classes e o esquema de coleções em diagramas separados. A Figura 4 apresenta o esquema de classes correspondente ao diagrama da Figura 2. O esquema de objetos correspondente é mostrado na Figura 5.

Confrontando-se as três figuras, conclui-se que, no diagrama geral da Figura 2, as três coleções, Empregado, Projeto e Departamento, definem implicitamente três classes correspondentes. Na verdade, neste diagrama, a notação das coleções com classes implicitamente definidas é uma abreviatura, conforme mostrado na Figura 6.

O Diagrama de Objetos da Figura 2 é um esquema que pode ser feito para a Companhia. De acordo com o método de modelagem adotado, outros esquemas poderiam ser criados para representar a mesma informação. Por exemplo, não há necessidade de se ter três coleções globais (Empregado, Projeto e Departamento), bastando existir uma delas. Como todos os objetos estarão ligados entre si, uma coleção persistente garante a persistência dos demais objetos.

Todavia, se só existir uma das coleções globais, todo acesso a qualquer informação será feito através desta coleção. Na modelagem adotada foi empregado um outro critério: toda entidade forte corresponde a uma coleção global. Com isto se tem acesso aos

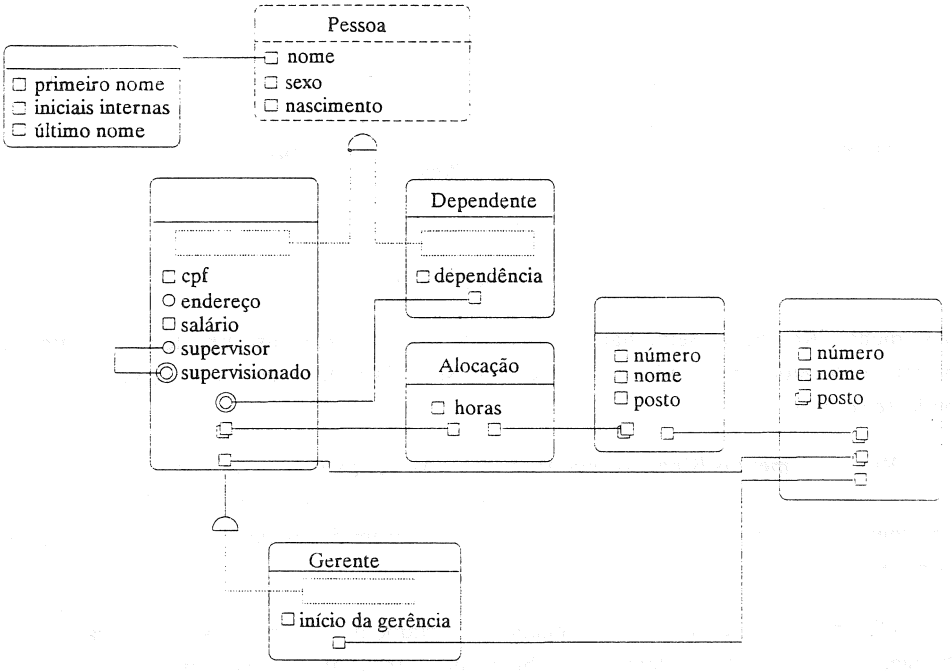


Figura 4: Esquema de Classes

empregados da companhia independentemente dos projetos e departamentos; se tem acesso aos projetos independentemente, e aos empregados também.

Representação de Associações Constantes e Variáveis - Os nomes Empregado, Projeto e Departamento, dentro do escopo da base de objetos da Companhia, estão associados a coleções. Estas são associações constantes. Isto significa que a coleção Empregado pode ter o seu conteúdo alterado, colocando-se ou retirando-se empregados, porém o nome Empregado estará constantemente associado a esta mesma coleção. Outras associações constantes envolvem os nomes das classes Pessoa, Dependente, Alocação e Gerente.

Já os atributos foram todos modelados usando-se associações variáveis. Por exemplo, todo empregado tem um salário, porém este pode ser alterado. A regra para representar associações variáveis e constantes nos Diagramas de Objetos é a seguinte: uma associação constante de um nome a uma entidade é feita colocando-se o nome dentro da representação da entidade; em uma associação variável, o nome fica fora da representação da entidade.

Alguns atributos poderiam ser modelados usando associações constantes, como na Figura 7. Neste diagrama, não é permitido que um empregado mude o seu CPF. Também não é permitido que uma pessoa na companhia mude de sexo ou de data de

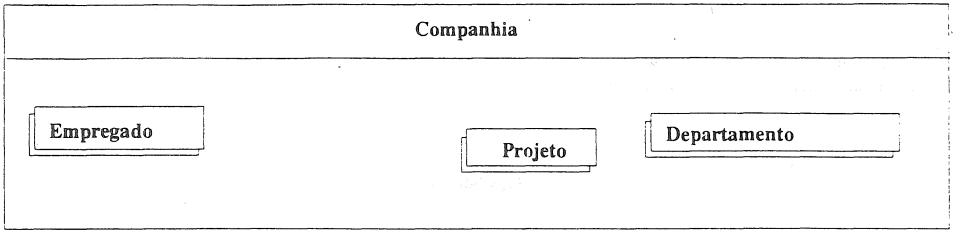


Figura 5: Esquema de Objetos

nascimento. Restrições semelhantes poderiam ser colocadas caso um departamento ou projeto não pudessem mudar de número ou se a data de início de uma gerência não pudesse ser alterada.

Relacionamentos Binários sem Atributos - As ligações entre as entidades representam os tipos de relacionamentos existentes entre elas. Assim vê-se, na Figura 2, que empregados e departamentos se relacionam. As extremidades das ligações representam os mapeamentos.

Observando-se a ligação Empregado-Departamento, interpreta-se que, através desta ligação, um empregado tem um atributo que representa um mapeamento em um departamento. Implicitamente, o nome deste mapeamento é "departamento". Portanto, dado um empregado pode-se falar no "departamento do empregado".

O mapeamento inverso também está especificado. Um departamento tem um atributo, com o nome "empregado" implícito, que representa uma coleção de empregados. Um objeto percebe o seu relacionamento com outro(s) objeto(s) através de um atributo equivalente ao mapeamento do objeto no(s) outros(s). A representação gráfica adotada segue a linha de representar os relacionamentos com uma visão orientada a objetos.

Outros Relacionamentos - Como mencionado anteriormente, relacionamentos não binários ou com atributos têm que ser tratados como objetos em si. Isto é exemplificado, na Figura 2, pela ligação Empregado-Projeto através da classe Alocação. Um empregado pode trabalhar em vários projetos e um projeto pode ter vários empregados a ele alocados. Isto explica os atributos em Empregado e Projeto. Entretanto, cada alocação de um empregado a um projeto precisa guardar um atributo contendo o número de horas que o empregado está alocado a este projeto.

Modelando uma alocação como sendo um tipo de objeto, conforme a Figura 2, um empregado tem um conjunto de alocações a projetos e um projeto tem um conjunto de alocações de empregados. Neste diagrama, o relacionamento Empregado-Departamento foi partido em dois relacionamentos: Empregado-Alocação e Projeto-Alocação. O defeito desta quebra é visível quando se nota que nada restringe um empregado a ter

duas alocações para um mesmo projeto. A Figura 7 apresenta uma notação que, não partindo a ligação Empregado-Projeto, soluciona esta questão.

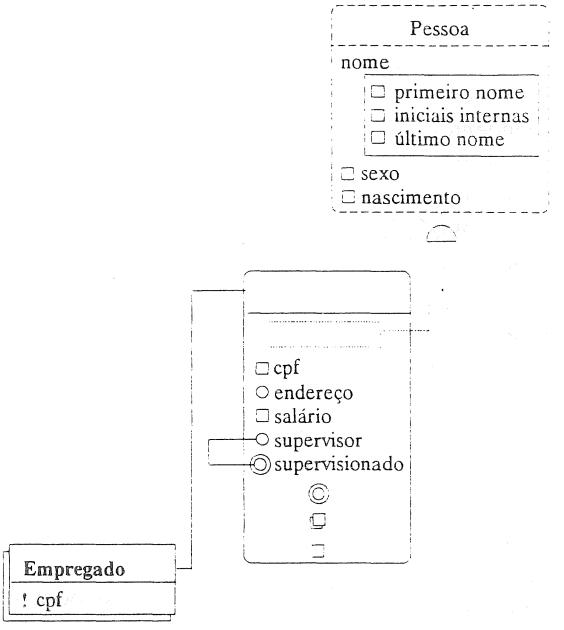


Figura 6: Expansão da Coleção Empregado

Mapeamento Sem Inverso - Pela Figura 2 observa-se que empregados se relacionam com objetos da classe Dependente. A representação utilizada para o atributo de Empregado correspondente ao seu mapeamento em dependentes significa que o valor deste atributo é uma coleção possivelmente vazia de objetos da classe Dependente.

Inversamente, em Dependente há um atributo que indica de qual empregado a pessoa é dependente. Esta é uma opção da modelagem da base de objetos. Caso seja assumido que os dependentes só são manipulados a partir dos empregados, o último mapeamento não precisa ser especificado, como na Figura 8.

Nomes de Mapeamentos Explícitos - O auto-relacionamento Empregado-Empregado, indicando que um empregado pode (ou não) ter supervisor e pode (ou não) ter empregados supervisionados, é representado pelos atributos **supervisor** e **supervisionado**. Estes atributos tiveram que receber nomes explicitamente, ao contrário dos demais atributos de mapeamento. Isto foi necessário porque a regra de nomes implícitos estabelece que o nome de um mapeamento é o mesmo da classe ou objeto para o qual será feito o mapeamento. Assim, o atributo em Empregado que mapeia para os seus dependentes se chama **dependente**, pois mapeia para objetos da classe Dependente. Se este nome não fosse adequado para explicar o mapeamento, o projetista poderia especificar um outro explicitamente. Para evitar ambiguidade, o auto-relacionamento não pode usar nomes implícitos.

Classes Implícitas - Os atributos **cpf**, **endereço** e **salário** de Empregado também representam mapeamentos. Nos diagramas não foram empregadas ligações para ligar estes atributos às classes de seus elementos porque, no projeto, estas classes foram consideradas triviais e predefinidas (texto, número, data, etc.).

Herança e Pseudoclasses - Pelo DER da Figura 1 pode-se verificar que tanto empregados como seus dependentes possuem atributos comuns: nome, sexo e nascimento. Nos Diagramas de Objetos esses atributos poderiam ser colocados em evidência com a criação de uma superclasse para empregados e dependentes. Isto está representado na Figura 8, com a criação da classe Pessoa. Tanto Empregado como Dependente são Pessoa e herdam suas propriedades. Como não existe, nesta base, uma pessoa que não seja um empregado ou um dependente, Pessoa é uma pseudoclasse e assim foi representada. Note-se também que os nomes de dependentes passaram a ser vistos do mesmo modo que os nomes de empregados: estruturados em **primeiro nome**, **iniciais internas** e **último nome**. Em um DER convencional, sem definição de classes, tipos ou domínios, esta possibilidade aumentaria a complexidade do diagrama resultante.

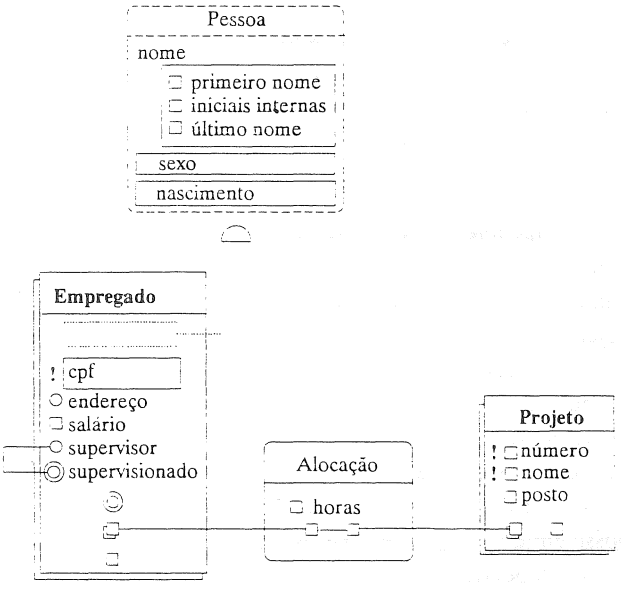


Figura 7: Associações Constantes em Atributos e Ligação Empregado-Projeto através de Alocação

A estruturação de **nome** de Pessoa é realizada dentro da própria caixinha que designa o atributo. Isto é feito assim por comodidade. A classe que descreve a estrutura dos nomes de pessoas é derivável automaticamente deste diagrama, como pode ser observado na Figura 4. Uma modelagem alternativa criaria explicitamente uma classe para nomes de pessoas e faria uma ligação do atributo **nome** de Pessoa a essa classe.

Alguns empregados gerenciam departamentos. Este relacionamento parcial fica melhor modelado com a definição da subclasse Gerente que representa os empregados envolvidos neste relacionamento. A data de início da gerência passa a ser um atributo exclusivo desse tipo de empregado.

Sobreposição - Nada impede, no DER da Figura 1, que um gerente trabalhe em um departamento mas gerencie outro. O Diagrama de Objetos apresentado proíbe esta situação usando o fato que uma subclasse pode alterar a definição de alguma propriedade que ela tenha herdado de suas superclasses. Isto é feito do seguinte modo: o relacionamento Empregado-Departamento definiu implicitamente em Empregado um atributo

departamento que indica qual departamento o empregado trabalha. Um gerente ao se relacionar com o departamento que gerencia, na ligação Gerente-Empregado, redefine o atributo **departamento**, pois não foi dado um outro nome explicitamente para este mapeamento. Portanto, o departamento de um gerente é o que está envolvido no relacionamento de gerência. Esta característica permite uma representação elegante deste tipo de restrição.

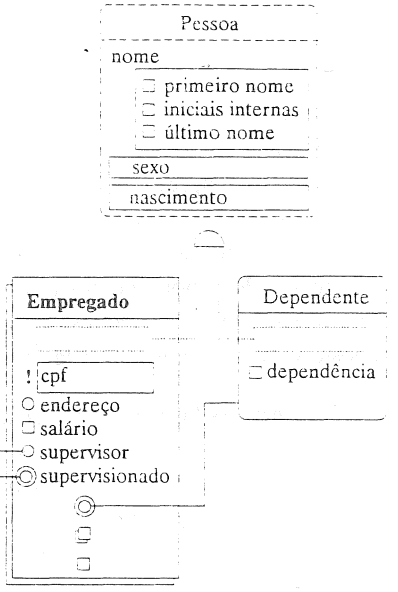


Figura 8: Mapeamento sem Inversa

Restrições de Chave - O exemplo define chaves candidatas para as coleções Empregado (cpf), Projeto (número e nome) e Departamento (número e nome). Os Diagramas de Objetos representam os atributos chaves precedendo-os por um sinal de exclamação (!).

A restrição de que, para um mesmo empregado, não há dependentes com o mesmo nome é definida indicando que a coleção de dependentes de cada empregado, representada pelo atributo de nome implícito **dependente**, tem **nome** como chave. Entretanto, esta restrição deve ser colocada apenas na coleção referente ao atributo **dependente**, como na Figura 9.

4. Considerações Finais

Sumário - Este trabalho apresentou os Diagramas de Objetos, usáveis na modelagem conceitual de bases de objetos do GEOTABA. A propriedade fundamental dos Diagramas de Objetos é atender ao mesmo tempo a representação de modelagens semânticas de informação e serem mapeados trivialmente no Modelo de Objetos do GEOTABA.

Os diagramas espelham diversos conceitos do Modelo de Objetos, tais como: objetos, classes e coleções; associações constantes e variáveis; reconhecimento da existência de relacionamentos entre objetos e representação destes relacionamentos através de seus mapeamentos; atributos e mapeamentos opcionais e obrigatórios, univalorados e multivalorados; herança e sobreposição de propriedades através de hierarquia de classes; chaves de coleções.

O foco desta apresentação foi dado em como a semântica da informação contida em uma modelagem tradicional, utilizando um DER comum, pode ser representada pelos Diagramas de Objetos. Isto foi mostrado através de um Diagrama de Objetos que mo-

dela a informação descrita em um DER utilizado didaticamente para exemplificar as possibilidades dos Diagramas de Entidades-Relacionamentos. Também foi mostrado que as noções empregadas neste Diagrama de Objetos exemplo são capazes de modelar propriedades não representáveis pelos Diagramas de Entidades-Relacionamentos tradicionais, tais como, associações constantes, herança e sobreposição de atributos e de relacionamentos.

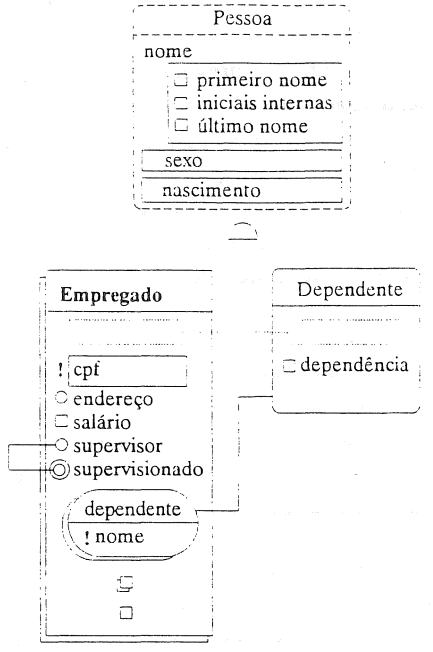


Figura 9: Chave Parcial

Perspectivas Futuras - Os Diagramas de Objetos, conforme mostrados neste trabalho, servem como base para extensões que introduzam noções adequadas a capturar outros tipos de restrições diferentes das tratadas no exemplo utilizado. Especificamente, este é o caso da junção e disjunção de ligações, com a finalidade de se estabelecer condições de integridade que envolvam mais de um relacionamento.

Uma série de conceitos definidos no Modelo de Objetos do GEOTABA não foram empregados neste exemplo e, consequentemente, não tiveram suas representações introduzidas nos Diagramas de Objetos deste trabalho. Este é o caso da especificação de atributos componentes, da especificação de cardinalidade mínima e máxima pouco usuais, a utilização de objetos e classes pré-existentes ou padrões, e da especificação do protocolo de serviços realizáveis pelos objetos.

Mais além, os Diagramas de Objetos devem ser encarados como um subconjunto de uma linguagem visual de definição e manipulação de objetos do GEOTABA. Para isto ser alcançado, deve ser possível especificar interativamente processamentos e executá-los. Os passos seguintes serão fornecer a possibilidade de se representar não apenas o nível conceitual, mas também os demais níveis de abstração do Modelo de Objetos: nível externo, de representação e de implementação. Isto inclui a especificação dos mapeamentos entre os diferentes níveis de abstração.

Conclusões - Os Diagramas de Objetos têm sido empregados na especificação do metaesquema do GEOTABA e como ferramenta para ajudar a compreensão e o aperfeiçoamento do seu Modelo de Objetos. Como os conceitos deste modelo são representados graficamente nos diagramas, as pessoas encontram mais facilidade de fixar as noções envolvidas e seus propósitos.

Como etapa preliminar para a especificação do metaesquema do GEOTABA, foi elaborado um Diagrama de Objetos representando o metaesquema do sistema Smalltalk/V [DIGITALK 86]. Por metaesquema, entenda-se o esquema de classes e objetos predefinidos envolvidos na criação e manipulação básicas de classes e objetos, ou seja, as classes **Object, Class, Metaclass, Behavior, CompiledMethod**, etc. Algumas extensões foram necessárias para se poder representar serviços prestados por estas classes (compilar um texto fonte, criar um objeto, executar uma mensagem, etc.) e para representar restrições não convencionais.

Embora não tenha sido desenvolvido um editor de Diagramas de Objetos, dois fatos permitem prever que não haverá dificuldades maiores nesta implementação. O primeiro deles é a trivialidade do mapeamento da representação gráfica nos Diagramas de Objetos em classes e objetos do GEOTABA. O segundo fato é a experiência na realização de um editor de diagramas semelhantes [MATTOSO 90].

5. Referências

[ALABISO 88] B. ALABISO, **Transformations of Data Flow Analysis Models to Object-Oriented Design**, in *Proceedings of OOPSLA'88*, ACM, New York, pp. 335-353, 1988.

[BOOCH 91] GRADY BOOCH, *Object-Oriented Design with Applications*. The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1991.

[CHEN 76] P.P. CHEN, **The Entity-Relationship Model - Toward a Unified View of Data.** *ACM Trans. Database Syst.* 1:1, pp. 9-36, março 1976.

[COAD 90] PETER COAD, EDWARD YOURDON, *Object-Oriented Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1990.

[DAYAL 84] U. DAYAL, H.Y. HWANG, **View Definition and Generalization for Database Integration in a Multidatabase System.** *IEEE Trans. Soft. Eng.* **SE-10**:6, pp. 628-644, 1984.

[DIGITALK 86] *Smalltalk/V: Tutorial and Programming Handbook*, Digitalk Inc., Los Angeles, Calif., 1986.

[ELMASRI 89] RAMEZ ELMASRI, SHAMKANT B. NAVATHE, *Fundamentals of Database Systems*. The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1989.

[HENDERSON 90] BRIAN HENDERSON-SELLERS, JULIAN M. EDWARDS, **The Object-Oriented Systems Life Cycle.** *Communications of ACM* 33:9, pp. 142-159, setembro de 1990.

[HULL 87] RICHARD HULL, ROGER KING, **Semantic Database Modeling: Survey, Applications, and Research Issues,** *ACM Computing Surveys* 19:3, pp.201-260, Setembro 1987.

[MATTOSO 90] ADRIANA L.Q. MATTOSO, *TABA_OBJ: Um Ambiente de Desenvolvimento de Software com Orientação a Objetos.* Tese de M.Sc. em Eng. de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, Agosto de 1990.

[MONTE 90] LUIZ C.M. MONTE, *O Modelo de Objetos do GEOTABA.* Exame de Qualificação para doutoramento em Eng. de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, 1990.

[SHIPMAN 81] D. SHIPMAN, **The Functional Data Model and the Data Language Daplex.** *ACM Trans. Database Syst.* 6:1, pp. 140-173, março 1981.

[TEOREY 86] TOBY J. TEOREY, DONGQING YANG, JAMES P. FRY, **A Logical Design Methodology for Relational Databases using the Extended Entity-Relationship Model,** *ACM Computing Surveys* 18:2, pp. 197-222, Junho 1986.

Projeto Conceitual e Lógico de Sistemas de Gerência de Banco de Dados Orientado a Objetos

Cláudio N. F. Trotta, Cláudia M. L. Werner

UFRJ/COPPE Sistemas
Cidade Universitária - CT - sala H-319
Caixa Postal 68511 - CEP 21945 - Rio de Janeiro - RJ
Tel. (021) 590-2552/2652

Sumário

Neste trabalho fazemos uma análise do emprego do Modelo de Entidades e Relacionamentos como ferramenta conceitual para a fase anterior ao projeto lógico de um banco de dados orientado a objetos. Abordamos a mudança de perspectiva necessária para que possamos utilizá-lo com sucesso neste novo contexto, inclusive introduzindo a abstração conceitual de especialização de relacionamento. Apresentamos, ainda, considerações para o projeto lógico, específicas ao SGBDOO utilizado.

1. Introdução.

O Modelo de Entidades e Relacionamentos (MER) [CHEN 76] foi concebido, e tem sido utilizado, como ferramenta conceitual para o projeto de banco de dados relacional. Na tentativa de utilizar esta ferramenta para reprojetar uma aplicação, anteriormente implementada no modelo relacional, seguindo o paradigma da orientação a objetos, notamos que o esquema lógico final se distanciou muito do Diagrama de Entidades e Relacionamentos (DER) obtido anteriormente. Este artigo tem como objetivo apontar as razões pelas quais isto ocorre, oferecendo recomendações para o uso do MER neste contexto.

Outra questão considerada é o processo do projeto lógico, isto é, a passagem do DER para o esquema de classes de um Sistema de Gerência de Banco de Dados Orientado a Objetos (SGBDOO). Neste processo, como ainda não existe uma definição consensual sobre os requisitos de um SGBDOO, soluções específicas foram tomadas para o sistema usado, no caso, o O2.

O O2 é um SGBDOO em desenvolvimento num projeto de cinco anos do consórcio Altair, França, iniciado em 1986. O sistema provê um ambiente multi-língue de programação (ex. CO2, BasicO2), uma linguagem de consulta, uma interface alfanumérica, um ambiente de programação (OOPE) e um conjunto de ferramentas geradoras de interface (LOOKS) [DEUX 89]. Ao interagir com o O2, o usuário pode escolher entre utilizar a interface alfanumérica ou o ambiente OOPE, sendo esta última a interface mais usada, pois oferece ferramentas gráficas para atualização, consulta e navegação na base de dados. O sistema utilizado é o protótipo, versão 1.1, distribuido para avaliação, que roda em equipamentos Sun 3/60, 3/80 ou 3/280S, com sistema operacional SunOS versão 4.0.

Na próxima seção descrevemos o projeto conceitual da aplicação. Na seção três apresentamos o modelo de dados do O2, com o objetivo de fornecer informações necessárias para o projeto lógico, descrito na seção quatro.

2. A Modelagem da Aplicação.

A aplicação reprojetada é um protótipo de um dicionário de dados automatizado, uma das ferramentas encontradas num ambiente do tipo **FEGRES** [TRAVASSOS 90]. Apesar desta aplicação já ter sido, previamente, modelada em [TROTTA 90a, 90b] e, efetivamente, construída usando-se um banco de dados relacional, o enfoque da orientação a objetos, agora abordado, nos levou a um esquema conceitual diferente do anterior.

Esta aparente contradição é explicada, se abandonarmos a visão ingênua de que o poder de expressão do método não determina, de forma direta, o esquema conceitual a ser produzido. Anteriormente, utilizou-se o MER como ferramenta conceitual e o Modelo Relacional como ferramenta para a implementação. Agora, embora utilizemos o MER como uma ferramenta gráfica de comunicação e modelagem, o paradigma da orientação a objetos dita as abstrações conceituais que serão reconhecidas no processo de modelagem e o modelo de dados específico do O2 especializa mais ainda estas abstrações. Desta forma, embora os construtores básicos das abstrações conceituais do DER, construído anteriormente [TROTTA 90b], sejam os mesmos do DER agora apresentado (figura 1), o esquema conceitual produzido é diferente, graças a esta alteração da perspectiva. O diagrama apresentados na figura 1 segue a convenção do MER extendido apresentada em [ELMASRI 89], com exceção da abstração de **especialização de relacionamento** que será apresentada na seção quatro. O esquema completo do O2 correspondente à figura 1 encontra-se no apêndice.

Podemos dizer que as diferenças entre os dois esquemas são causadas pelos seguintes fatores:

1) O relativamente forte poder de expressão do MER na parte estrutural em comparação com a capacidade de representar operações (nenhuma) e restrições de integridade (limitada);

2) A eliminação do conceito de chave primária.

A seguir discutimos cada um destes fatores.

2.1.Poder de expressão do MER.

No MER (quando utilizado como ferramenta conceitual para o modelo relacional) diversas restrições de integridade são capturadas, criando-se entidades e relacionamentos adicionais. Isto é, como o modelo é "forte" na parte estrutural, as restrições de integridade são, algumas vezes, "transformadas" em estruturas de dados não

naturais. É o exemplo de algumas categorias criadas no diagrama anterior (Rep_Nó, Rep_Construtor e Construtor) [TROTTA 90b]. As restrições de integridade que estas estruturas representam são, detalhadamente, explicadas no mesmo trabalho.

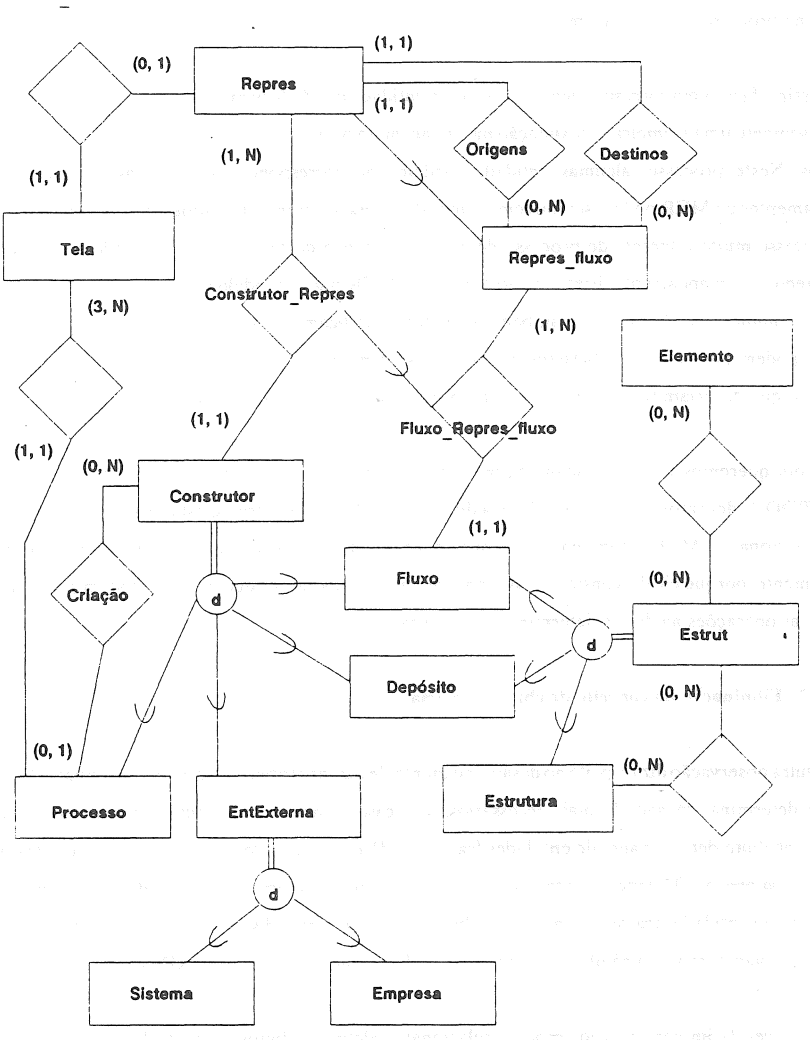


Figura 1 - O Esquema para o O2.

Ao passar para uma representação orientada a objetos, é possível modelar também o comportamento. Este aspecto, normalmente, desprezado totalmente no MER, pode levar ao aparecimento de hierarquias de generalização/especialização criadas somente com o intuito de fatorar operações. Além disto, as restrições de

integridade podem, também, ser incorporadas às especificações das operações. A combinação destes dois fatores, isto é, a criação de hierarquias de generalização/especialização para fatorar operações que mantêm a integridade, foi um ponto muito importante nesta aplicação específica (vide 4.2). Estes dois fatores influiram bastante no processo de modelagem.

Assim, fazer uma correspondência 1:1 de uma entidade do MER para uma classe na orientação a objetos, deve ser somente uma primeira aproximação na passagem do projeto conceitual para o projeto lógico orientado a objetos. Neste processo, algumas entidades podem não corresponder a uma classe e vice-versa, e os relacionamentos no MER podem ser implementados de formas diferentes no esquema orientado a objetos. Esse é um processo muito diferente do processo de projeto de banco de dados relacional, onde o projeto lógico é, praticamente, um mapeamento direto das abstrações do MER para o modelo relacional. No caso da orientação a objetos, o número de alterações e, também, sua natureza indicam novas abstrações (ex. duas entidades, antes distintas, podem passar a ser subclasses de uma classe comum, devido ao interesse em, também, fatorar as operações) que deveriam ter sido detectadas no projeto conceitual e não no lógico.

O que queremos dizer é que ao utilizarmos o MER para representar uma aplicação que será implementada num SGBDOO, devemos abordar o problema diferentemente do que seria se esta fosse implementada um em SGBD relacional. O MER, assim, não é de um nível conceitual tão elevado quanto gostaríamos. A razão disto é, basicamente, porque ele desconsidera, sistematicamente, as operações que os dados podem sofrer. Em outras palavras: as operações ajudam a determinar a estrutura dos dados.

2.2. Eliminação do conceito de chave primária.

Outra observação diz respeito ao desaparecimento de alguns atributos. No nível conceitual (MER), alguns atributos determinantes não são mais necessários. É o caso, também, de atributos não naturais criados para compor o atributo determinante de entidades fracas do MER. Isto acontece pois estas entidades fracas podem passar a ser *valores* no O2 (vide seção três) da entidade forte que as determinam, ou ainda, mesmo que continuem existindo como entidades fracas, o conceito de objeto é suficiente para distinguir suas instâncias. Isto vem sendo discutido amplamente sob o rótulo de "problema de identidade do objeto" [ATKINSON 89].

No nível da implementação (modelo relacional), alguns atributos de controle (como, por exemplo, a identificação da última instância de representação gráfica criada, necessária para compor a chave da próxima instância a ser criada) também se tornam desnecessários, pois objetos podem ser identificados independente destes atributos.

3. O Modelo de Dados do O2

No modelo de dados do O2 existe uma distinção entre os chamados *valores* e *objetos*. Objetos em O2, como em linguagens de programação orientadas a objetos (OOPL) (ex. Smalltalk, C++, etc), possuem uma identidade e encapsulam dados e comportamento (i.e. métodos). Valores, entretanto, podem ser vistos como dados, simplesmente, não obedecendo aos princípios da orientação a objetos da seguinte forma [LÉCLUSE 89a]:

- não possuem identidade;
- sua estrutura, embora similar a de objetos, é visível e manipulável;
- a manipulação é feita através de operadores primitivos, não por métodos.

Valores foram introduzidos em O2 para permitir a manipulação de estruturas complexas de dados, não existentes em OOPLs. Eles podem ser do tipo *atômico* (ex. inteiro, real, booleano, etc) ou *estruturado* (ex. tuple, set, list).

Objetos pertencem a *classes*. Classes descrevem a estrutura e o comportamento de uma coleção de objetos. A parte estrutural é um *tipo,* e a parte comportamental é um conjunto de *métodos* . A definição de uma classe consiste em prover : seu nome, seu tipo e seus métodos. É possível definir, neste instante, somente os nomes dos métodos para, posteriormente, definirmos seu código.

Todas as classes em O2 são subclasses diretas ou indiretas de uma classe pré-definida, denominada OBJECT. Um esquema em O2 é um conjunto de classes ligadas pela relação de herança, onde herança múltipla é permitida desde que não haja conflitos de nomes.

Um objeto é uma *instância* de uma classe e pode ser visto como um par *(identificador,valor)*. Como uma classe em O2, por "default", não é persistente, um objeto só será persistente se for referenciado por objetos persistentes ou nomeados. Para que os objetos de uma classe sejam persistentes é preciso definí-la com a cláusula **with extension**, explicitamente. Na versão do protótipo usado, entretanto, esta cláusula não está implementada, tendo sido necessário criarmos valores nomeados para os "sets" de objetos das classes que deveriam ser persistentes.

Métodos são escritos utilizando extensões de linguagens de programação tradicionais (ex. CO2, BasicO2), sendo que o protótipo usado oferece apenas a linguagem CO2. A definição de um método consiste em fornecer sua assinatura (i.e., seu nome, parâmetros, tipos ou classes dos parâmetros e tipo ou classe de seu resultado) e seu corpo (i.e., código em co2). Métodos podem, ainda, ser *públicos* (podem ser chamados a partir de quaisquer classes) ou *privados* (só podem ser chamados por métodos na mesma classe).

Uma estrutura de dados de um classe pode ser *privada* ou *pública*. A estrutura pública oferece um

relaxamento no encapsulamento dos dados, permitindo que a estrutura seja acessada por métodos de outras classes.

4. O Projeto da Aplicação.

No projeto lógico da aplicação tivemos que considerar o paradigma da orientação a objetos com as limitações específicas do SGBD usado, no caso o O2. O maior problema encontrado foi como representar relacionamentos pois não há uma representação direta para esta abstração. Além disto, existe uma certa assimetria na parte estrutural e na linguagem de consultas que causaram algumas dificuldades adicionais. A seguir descrevemos, detalhadamente, este problema e duas formas encontradas para contorná-lo.

4.1. Assimetria do modelo de dado do O2.

A assimetria do O2 em representar relacionamentos, combinada com a forma de expressar consultas, levam a formas alternativas de modelagem. Exemplificando, um relacionamento de cardinalidade 1:N, entre objetos de duas classes A e B, pode ser representado:

i) "à moda relacional", isto é, colocando-se um ponteiro para a classe "do lado do 1", na classe "do lado do N":

```
add class A
type tuple (.....);

add class B
type tuple (.....,
            a: A,
            ....);
```

ii) usando-se o construtor de set:

```
add class A
type tuple (....,
            b: set(B),
            ...);

add class B
type tuple (.....);
```

iii) ou ainda, usando-se uma combinação de i e ii, obtendo-se uma representação simétrica, mas com controle de integridade sob responsabilidade do usuário/programador.

```
add class A
type tuple(....,
           b: set(B),
           ...);

add class B
type tuple (.....,
            a: A,
            ....);
```

A decisão sobre a alternativa preferível pode ser tomada em função da facilidade em programar os métodos em cada uma delas. É interessante notar que esta percepção só acontece num nível de detalhamento mais profundo, isto é, numa etapa muito posterior a que se deseja, ao projetar um esquema conceitual.

Para se verificar as implicações negativas de uma modelagem assimétrica, agravadas pelas construções permitidas na linguagem de consulta do O2, considere o seguinte exemplo:

```
add class Aluno
type tuple (nome:string,
            endereço:string,
            matrícula:set(Matrícula));

add class Matrícula
type tuple(nota:integer);

add class Curso
type tuple (código:integer,
            título:string,
            matrícula:set(Matrícula);
```

As classes acima representam um relacionamento N:M, de nome "Matrícula", entre as entidades *Aluno* e *Curso*. Como no projeto relacional, escolhemos representar este relacionamento como uma classe de nome *Matrícula*. Outras opções de modelagem seriam possíveis, por exemplo, colocando a nota diretamente nas classes *Aluno* e/ou *Curso*.

A simples consulta "listar os nomes dos alunos que cursam a disciplina cujo título é Cálculo", que em SQL seria, simplesmente:

```
Aluno   (nome,endereço)
Matrícula  (nome,código,nota)
Curso (código,título)

select  a.nome
from    a in Aluno, m in Matrícula, c in Curso
where  a.nome = m.nome and c.código =m.código
         and c.título = "Cálculo";
```

No O2 ficaria:

```
select   a.nome
from     a in Aluno,
         m in a.matrícula
where    m is in  (  select m1
                     from   c in Curso,
                            m1 in c.matrícula
                     where c.título = "Cálculo");
```

A dificuldade é causada pela inexistência de uma operação equivalente a operação relacional de junção Se a classe *Matrícula* for definida de forma a tornar o modelo simétrico, teríamos:

```
add class Matrícula
type tuple(nota:integer,
           aluno:set(Aluno),
           curso:set(Curso));

select   a.nome
from     a in Aluno,
         m in a.matrícula,
         c in m.curso
where  c.título = "Cálculo";
```

ou

```
select   a.nome
from     c in curso,
         m in c.matrícula,
          a in m.aluno
where    c.título = "Cálculo";
```

Embora a estrutura de dados tenha se tornado simétrica, as consultas ainda lembram uma navegação hierárquica no sentido *aluno-matrícula-curso* ou *curso-matrícula-aluno*, respectivamente.

4.2. Resolvendo a assimetria na aplicação.

Na aplicação do Dicionário de Dados escolhemos testar duas formas de resolver o problema da assimetria na modelagem de relacionamentos entre classes:

i) Mantendo simétrica a estrutura que representa os relacionamentos e controlando as restrições de integridade decorrentes da redundância introduzida; ou

ii) Representando a estrutura assimetricamente e enfrentando o problema da assimetria da linguagem de consulta.

No projeto lógico, mantivemos a representação simétrica em todos os relacionamentos, exceto em dois deles, escolhidos aleatoriamente.

4.2.1. Projeto Simétrico.

No primeiro caso, as restrições de integridade foram testadas valendo-se de operações de alto nível disponíveis na linguagem CO2. Vejamos o caso do relacionamento 1:N *Criação* (vide figura 1). Este relacionamento é representado pelos atributos *criador* da classe *Construtor* e *filhos* da classe *Processo*.

```
add class  Construtor
        inherits object
        type tuple (criador:Processo,
                    rep: set (Repres));

add class Processo
        inherits Construtor
        type tuple (ident_p:string;
                    nome:string,....
                    filhos:set(Construtor));

add name Processos: set (Processo);
```

A última linha cria um valor nomeado *Processos*. Todo objeto da classe *Processo* que for inserido neste "set" será persistente.

A' interface LOOKS [O2 89c], fornecida com o sistema, permite manipular os objetos na tela sem necessidade de programação. Por exemplo, para colocarmos um *Processo* como o criador de um *Construtor*, basta arrastarmos pela tela o ícone que representa o *Processo*, parando em cima da caixa de diálogo que representa o atributo *criador* (do *Construtor*).

Se o usuário realizar tal movimento, torna-se necessário manter a integridade da base de dados, que foi ferida, pois os dados devem ser simétricos. Em outras palavras, se o processo P1 foi arrastado para ser o criador do fluxo F1, é necessário incluir o fluxo F1 no conjunto de filhos de P1.

Um método de nome *integr_criador* da classe *Construtor* é responsável por esta restrição:

```
add method integr_criador in class Construtor
body integr_criador (paiant:Processo):object in co2
{
if (self->criador != paiant)
        {
        if (paiant != nil)
           {paiant->filhos -= set (self);};
        if (self->criador != nil)
           {self->criador->filhos += set (self);};
        };
}
```

A variável *paiant* é passada como parâmetro e representa o valor anterior do atributo *criador* da classe *Construtor*. Se houver mudança na paternidade, o objeto em questão é incluído (ou retirado) do conjunto de filhos do seu novo pai (antigo pai).

Por outro lado, ao alterar-se um processo, é necessário, também, testar a variação no seu conjunto de filhos. Isto ocorre no caso em que o fluxo F1 teria sido arrastado para dentro do conjunto de filhos de P1. O seguinte trecho de programa, no método que modifica um processo, é responsável por isto:

```
O2 set(Construtor) filhosant, filhosnovos,filhosdeletados;
O2 Construtor p;
O2 Processo paiant;

filhosant = self->filhos;
paiant    = self->criador;

/* -- modificação dos dados do processo */

filhosnovos = self->filhos - filhosant;
filhosdeletados = filhosant - self->filhos;

for (p in filhosnovos) {p->criador = self};
for (p in filhosdeletados) {p->criador = nil};

[self integr_criador (paiant)];
```

Repare na operação de diferença de conjuntos e ao envio da mensagem *integr_criador* na última linha. A razão disto é porque *Processo* também **é-um** *Construtor* (veja no esquema).

4.2.2. Projeto Assimétrico.

Para a segunda solução, consideramos três relacionamentos representados assimetricamente no esquema lógico (refira-se à figura 1 e ao esquema no apêndice): *Fluxo_Repres_fluxo*; *Origens* e *Destinos*.

A entidade *Repres* significa uma representação gráfica no DFD de um *Construtor* (i.e., processo, depósito ou entidade externa), enquanto *Repres_fluxo* (especialização de *Repres*) representa uma representação gráfica no DFD de um fluxo de dados. Assim, o relacionamento *Fluxo_Repres_fluxo* é uma ***especialização do relacionamento*** *Construtor_Repres*. A figura 1 mostra uma representação gráfica para esta abstração. O significado disto é que uma especialização da entidade *Construtor* (a entidade *Fluxo*) não participa do relacionamento *Construtor_Repres*, mas sim do relacionamento *Fluxo_Repres_fluxo*.

Para a abstração de especialização de relacionamento fazer sentido, é necessário que aconteça entre relacionamentos cujas entidades participantes são especializações entre si, isto é, se o relacionamento

Fluxo_Repres_fluxo acontece entre as entidades *Fluxo* e *Repres_fluxo* e, ainda, ele é uma especialização do relacionamento *Construtor_Repres*, que acontece entre *Construtor* e *Repres*, então *Fluxo* é uma especialização de *Construtor* e *Repres_fluxo* é uma especialização de *Repres*.

Os relacionamentos *Origens* e *Destinos* do DER representam as ligações gráficas, isto é, contêm a informação de quais fluxos ligam quais construtores.

A definição do esquema é:

```
add class Fluxo
        inherits Construtor, Estrut
        type tuple (redefines rep:set (Repres_fluxo),
                    nome_fluxo:string,
                    descrição:string,..... );

add name Fluxos: set(Fluxo);

add class Repres
        inherits object
        type tuple (x:integer,
                    y:integer,
                    tela:Tela,
                   construtor:Construtor);

add class Repres_fluxo
        inherits Repres
        type tuple (tipo_traçado:integer,
                    origem: tuple (o:Repres,
                                   xo:integer,
                                   yo:integer),
                    destino: tuple (d:Repres,
                                    xd:integer,
                                    yd:integer));
```

Repare que na classe *Repres_fluxo* não existe um atributo para representar o relacionamento com a classe *Fluxo*, nem na classe *Repres* sequer existe um atributo para representar o relacionamento com *Repres_fluxo*.

É muito fácil navegar na direção dos relacionamentos. Por exemplo, com uma ficha de fluxo de dados na tela, podemos selecionar uma de suas representações (instância de *Repres_fluxo)* e, acionando o "mouse", teremos a opção (que aparece num cardápio) de consultar o construtor (i.e., processo, depósito ou entidade externa) donde se origina aquela representação. Como a navegação é no sentido *Repres_fluxo* - *Repres* - *Construtor*, e existem atributos representando os relacionamentos nesta mesma direção, o código em CO2 fica:

```
add method origens in class Repres_fluxo
body origens():object in co2
{...
self->origem.o->construtor}
```

Graças ao "desencapsulamento" permitido no modelo de dados, é possível atingir diretamente um *Construtor* a partir de um objeto da classe *Repres_fluxo* (self) navegando através dos atributos.

Porém, a recíproca, isto é, a partir da classe *Repres* atingir a classe *Fluxo* (por exemplo, dado um processo, queremos recuperar os fluxos que têm origem nele), é mais difícil de implementar em CO2. Seu código é:

```
add method origens in class Repres
body origens():object in co2

O2 Fluxo f;
O2 set(Fluxo) fs;
O2 Repres_fluxo w;

for (f in Fluxos)
        {aux = select w->origem.o
                from w in f->rep
                where w-origem.o == self end;

        if (aux != (O2 set (Repres)) set () )
                {fs += set(f);};
        };
/* fs é o conjunto de todos os fluxos que tem origem no construtor representado por self */
```

Na verdade, a linguagem de consulta permite expressar esta consulta num nível muito mais alto, porém, na atual versão do protótipo, somente um subconjunto desta linguagem pode ser embutida em CO2. Na linguagem de consulta , teríamos:

```
fs = select f
     from f in Fluxos
      where exists x in
                        (select w->origem.o
                        from w in f->rep): (x == self) end;
```

Dentro da concepção do modelo de dados do O2, onde valores devem ser usados para representar informação que não é compartilhada [O2 89a], as classes *Repres* e *Repres_fluxo* poderiam ter sido assim representadas. Porém, é importante que sejam, realmente, representadas como classes pois suas instâncias são manipuladas como objetos. Algumas das operações destes objetos são: movimentação na tela do DFD, consulta às origens e destinos, alteração de tamanho, etc. Isto reforça a idéia de que as operações ajudam a determinar a estrutura dos dados, conforme discutido em 2.1.

5. Conclusões.

Neste trabalho, indicamos que o maior problema na aplicação do MER, para o projeto lógico orientado a objetos, diz respeito a sua incapacidade de representar operações. Como vimos, as operações ajudam a determinar a estrutura dos dados. O problema por trás da questão é um velho costume de encarar os dados independente das operações e restrições inerentes a ele. Modelar um conceito do mundo real como um atributo ou como uma entidade pode ser determinado, simplemente, verificando-se a existência de operações específicas sobre este conceito. Em caso afirmativo, ele merece um "status" de entidade. Outro exemplo no qual observamos que as operações podem determinar estruturas de dados são as hierarquias de generalização/especialização que surgem com o objetivo único de fatorar operações.

Em termos de projeto lógico orientado a objetos, ainda existe uma grande dependência do SGBDOO utilizado. No caso do O2, a falta de representação direta de relacionamentos causou dificuldades que tiveram que ser contornadas programando-se métodos para implementar restrições de integridade ou convivendo-se com o problema da assimetria da linguagem de consultas. Outros modelos de objetos [MONTE 91] permitem uma representação mais natural.

O sistema O2, principalmente as facilidades do LOOKS, mostrou-se bastante satisfatório para o tipo de aplicação implementada. A maioria do código escrito correspondeu às restrições de integridade necessárias para manter os relacionamentos simétricos, o que demonstra a necessidade da incorporação de algumas abstrações dos modelos semânticos de dados em Sistemas de Gerência de Banco de Dados Orientados a Objetos.

Referências Bibliográficas.

Manuais do O2:

[O2 90] "The O2 System User's Guide"
Version 1.1, Released 15 February 1990. Printing revision 1.3, 26 April 1990.

[O2 89a] "The O2 Programmer's Manual"
Version 1.0, Released 15 December 1989. Printing revision 1.2, 9 January 1990.

[O2 89b] "The CO2 Programmer's Manual"
Version 1.0, Released 15 December 1989. Printing revision 1.2, 9 January 1990.

[O2 89c] "The LOOKS User's Manual"
Version 1.0, Released 15 December 1989. Printing revision 1.1, 9 January 1990.

[O2 89d] "The LOOKS Programmer's Manual"
Version 1.0, Released 15 December 1989. Printing revision 1.1, 9 January 1990.

[O2 89e] "OOPE: The Object-Oriented Programming Environment"
Version 1.0, Released 15 December 1989. Printing revision 1.1, 9 January 1990.

[O2 89f] "The O2 Query Language User's Manual"
Version 1.0, Released 15 December 1989. Printing revision 1.1, 9 January 1990.

Relatórios Técnicos:

[ATKINSON 89] M. Atkinson et al., "The Object-Oriented Database System Manifesto", Rapport Technique Altair 30-89.

[BANCILHON 88] François Bancilhon, "Object-Oriented Database Systems", Rapport Technique Altair 16-88.
[DEUX 89] O. Deaux at al., "The Story of O2", Rapport Technique Altair 37-89.
[LÉCLUSE 89a] C. Lécluse, P. Richard, "The O2 Data Model", Rapport Technique Altair 39-89.
[LÉCLUSE 89b] C. Lécluse, P. Richard, "The O2 DataBase Programming Language", Rapport Technique Altair 26-89.
[MONTE 91] L.C.M. Monte, "Uma Representação Gráfica para o Modelo de Objetos do GEOTABA", Relatório Técnico do Programa de Eng. de Sistemas e Computação ES-236/91, COPPE/UFRJ, Março 1991.

Artigos:

[CHEN 76] P.P. Chen, "The Entity-Relationship Model - Towards a Unified View of Data", ACM Transactions Data Base Systems, vol.1, no.1, March 1976.
[ELMASRI 89] R.Elmasri, S.B.Navathe, "Fundamentals of Database Systems", The Benjamin/Cummings Publishing Company, Inc. 1989.
[TRAVASSOS 90] G. H. Travassos, "FEGRES: FErramentas GRáficas para Engenharia de Software", Tese de Mestrado do Programa de Engenharia de Sistemas COPPE/UFRJ, 1990.
[TROTTA 90a] C..N.F. Trotta, G.H.Travassos, "Modelagem de Dados no Projeto de um Ambiente de Desenvolvimento de Software", anais da SUCESU 1990.
[TROTTA 90b] C.N.F. Trotta et al., "Modelagem de Dados de Aplicações não Convencionais: Um Estudo de Caso", anais do 4 Simpósio Brasileiro de Engenharia de Software, outubro 1990.

Apêndice

Esquema do Banco de Objetos O2

```
add class Construtor
/*      with extension*/
        inherits object
        type tuple(criador:Processo,
                rep: set(Repres));

public * in class Construtor;

add name Construtores : set(Construtor);

add class Repres
        inherits object
        type tuple (x:integer,
                y:integer,
                tela: Tela,
                construtor:Construtor);

public * in class Repres;

add class Repres_fluxo
        inherits Repres
        type tuple (tipo_tracado:integer,
                origem:  tuple(o:Repres,
                        xo:integer,
                        yo:integer),
                destino: tuple(d:Repres,
                        xd:integer,
                        yd:integer));

public * in class Repres_fluxo;
```

```
add class Processo
/*      with extension*/
        inherits Construtor
        type tuple (ident_p:string,
                nome:string,
                descricao:string,
                resumo_logico:string,
                refs_fisicas:string,
                det_logica:string,
                filhos:set(Construtor),
                tela: Tela);

public * in class Processo;

add name Processos: set (Processo);

add class Estrut
        inherits object
        type tuple (elementos: set(Elemento),
                estruturas: set(Estrutura));

public * in class Estrut;

add class Deposito
/*      with extension*/
        inherits  Construtor,Estrut
        type tuple (ident_d:string,
                nome:string,
                descricao:string,
                acesso_imediato:string,
                org_fisica:string);
```

```
public * in class Deposito;

add name Depositos: set (Deposito);

add class EntExterna
/*      with extension*/
        inherits Construtor
        type tuple (ident_e:string,
                    nome:string,
                    descricao: string);

public * in class EntExterna;

add name EntExternas: set (EntExterna);

add class Sistema
/*      with extension*/
        inherits EntExterna
        type tuple (linguagem:string,
                    hardware:string,
                    pessoapd:string,
                    fonepd:string);

public * in class Sistema;

add class Empresa
/*      with extension*/
        inherits EntExterna
        type tuple (setor:string,
                    pessoaorg:string,
                    cargo:string,
                    foneor:string);

public * in class Empresa;

add class Tela
/*      with extension*/
        inherits object
        type tuple (x:integer,
                    y:integer,
                    fundo:integer,
                    linhas:integer,
                    processo: Processo,
                    reps: set (Repres));

public * in class Tela;

add name Telas: set (Tela);

add class Elemento
        inherits object
        type tuple (nome_ele:string,
                    descricao:string,
                    tipoelemento:string,
                    dominio:string,
                    vertambem:string,
                    estruts: set(Estrut));

public * in class Elemento;

add name Elementos: set (Elemento);

add class Estrutura
        inherits Estrut
        type tuple(nome_est:string,
                   descricao:string,
                   volume:string,
                   estruts_pai: set(Estrut));

public * in class Estrutura;

add name Estruturas: set (Estrutura);

add class Fluxo
/*      with extension*/
        inherits  Construtor, Estrut
        type tuple (redefines rep: set(Repres_fluxo),
                    nome_fluxo:string,
                    descricao:string,
                    desc_ampliada:string,
                    volume:string,
                    tipo:integer);

public * in class Fluxo;

add name Fluxos: set (Fluxo);
```

CHOOSING AMONG UNCERTAINTY MANAGEMENT TECHNIQUES: A CASE STUDY

ALESSANDRO SAFFIOTTI† AND ELISABETH UMKEHRER‡

IRIDIA - Université Libre de Bruxelles
Av. F. Roosvelt, 50 - CP 194/6
1050 Bruxelles - Belgium
E-mail: r01507@bbrbfu01.bitnet

Abstract

We present an experimental approach to comparing uncertainty management techniques. The need for comparison arises from the observation that each of the many existing techniques may be adequate for a different type of problems. Our goal is to find the technique that "best" fits a given problem. We analyze two problems, one of academic interest, and one tailored on a real application. We formalize and solve both problems by probabilities, belief functions, possibilities, and Boolean values. Based on this experience, we discuss some intrinsic properties of the considered theories, as well as the adequacy of each theory to model the given problem. The experiment is performed by using PULCinella, a general tool for Propagating Uncertainty based on the Local Computation technique of Shafer and Shenoy. Pulcinella may be specialized to each of the four uncertainty theories above. Moreover, Pulcinella allows the user to define easily his own specializations.

† Partially supported by the ARCHON project, which is funded by grants from the Commission of the European Communities under the ESPRIT-II Program, P-2256.
‡ Supported by a grant of IRIDIA, Université Libre de Bruxelles.

1. Introduction

The need to represent and reason with uncertain information in Knowledge-Based systems has been generally recognized in AI. To cope with this need, researchers in this field have developed a number of formal theories and techniques for managing uncertainty (see Saffiotti, 1987, for a review). While this proliferation of techniques has lead some authors to try to identify one "best" theory of uncertainty (e.g. Cheeseman, 1985), it seems to us that a more promising perspective from the AI viewpoint is to see uncertainty theories as alternatives, among which we can choose the one that best fits our needs (Szolovits and Pauker, 1978; J. Fox, 1986; Clark, 1988). Our aim would then be to find the most adequate technique for the uncertainty type affecting our problem. In order to do this, the first step is to formalize each uncertainty technique in a common framework, and to express the proolem at hands in this framework. This step allows us to see how easily can our knowledge be expressed in each of the candidate techniques. As a possible outcome of this step, we may expect to discover that technique X requires data that is not available, or that technique Y cannot accommodate the data that is available. The second step is to compare the results obtained for our problem using the different techniques. Differences in the results may originate from different ways data has been coded in each technique, or from differences in the inference mechanism themselves.

In this paper, we propose using PULCinella, a general system for Propagating Uncertainty based on Local Computation, as a tool for testing and comparing uncertainty management techniques. Pulcinella is an implementation of the general framework for local computation proposed by Shafer and Shenoy (1988), and it is fully described in (Saffiotti and Umkehrer, 1991a). Accordingly, it may be instantiated to any of the theories which have been formalized in this framework. In particular, four specialization of Pulcinella have already been implemented, namely for propagating probabilities, belief functions, Boolean values, and possibilities. Moreover, Pulcinella makes it easy to implement new theories in it (provided that they can be modelled in Shafer and Shenoy's formalism). The key for Pulcinella's comparison power lies in the separation made between the process of modelling the structural knowledge of a problem, and that of modelling its qualitative knowledge. Once a structural model has been decided, we can superimpose any of the available uncertainty calculi on it. We will illustrate our comparison procedure using Pulcinella by analyzing two cases. In each case, the same problem will be formalized in all the four specializations discussed above. The first case deals with a

problem of academic interest: here, we will mainly concentrate on intrinsic differences between theories. The second case has been tailored on a real problem of diagnosing faults in electricity networks. Here, the knowledge engineer will find a discussion of the pros and cons of using different uncertainty theories for modelling the same test-bed problem.

The rest of this paper is organized as follows. Section 2 recalls some basic background on local computation, and presents the Pulcinella system. Section 3 and Section 4 present and discuss our two case studies. Section 5 concludes.

2. Pulcinella (Propagating Uncertainty by Local Computation)

2.1. A General Framework for Local Computation

Building on their work on belief function propagation (Shafer et al, 1987), Shenoy and Shafer (1988) propose a general framework for local computation. In this framework, the process of network propagation in itself has been abstracted from what is actually propagated. This framework has been further generalized by Shenoy (1989), who proposes a class of languages ("valuation-based languages") for building knowledge-based systems. These languages comprise objects, which are used to represent knowledge, and operators, which operate on these objects to make inferences on the knowledge. Two kinds of objects are considered, *variables* and *valuations*, and two operators, *combination* and *marginalization*[1]. We first remind the formal definitions of these elements, and will discuss their interpretation and use later.

Variables, Frames and Configurations. We consider a finite set of variables. Each variable may range over a finite set of possible values, called the *frame* for that variable. A *configuration* of a finite non-empty set of variables is an element of the cartesian product of the frames of the variables in this set.

Denotations: X for the complete set of variables; g, h, k for subsets of X;

W_g for the set of configurations of g; x,y for single configurations;

a,b,c for sets of configurations.

[1] A third operator, *solution*, is used for "decoding" the result obtained the propagation. This operator is not relevant to the present discussion, and so it will not be considered.

Sometimes we need to project a configuration of one set to another set. A configuration x of g is *projected* to h, $g \supset h$, by dropping all the elements in x belonging to g-h. It is *extended* to k, $k \supset g$, by building the cartesian product between the configuration and W_{k-g}.

Denotations: $x^{\downarrow h}$ for the projection of x to h; $x^{\uparrow h}$ for the extension of x to h.

Valuations. Given a set of variables h, we consider a set V_h. The elements of V_h are called *valuations* on the set h[2]. In our case, valuations are the objects that represent the uncertainty about a set of variables.

Denotations: V_g for the set of valuations on g; V for the set of all valuations on subsets of X; G, H for valuations.

Combination is any mapping $\otimes: V \times V \rightarrow V$, such that, if G and H are valuations on g and h, respectively, then $G \otimes H$ is a valuation on $g \cup h$.

Marginalization. For each $h \subseteq X$, there is a mapping $\downarrow h: \cup\{V_g \mid h \subseteq g\} \rightarrow V_h$, called *marginalization* to h, such that, if G is a valuation on g and $h \subseteq g$, then $G^{\downarrow h}$ is a valuation on h.

A set of variables, with their frames, together with a set of valuations, is called a *valuation system*. Once we have a valuation system, wa can evaluate it. This means to compute a global valuation on X obtained by combining together all the valuations in our valuation system, and find the marginals of this global valuation to each variable in X. Computing explicitly the global valuation is often unfeasible from the computational viewpoint. However, Shafer and Shenoy (1988) proposed a general local computation schema for evaluating valuation systems. The computation is local in the sense that combinations of valuations can be performed without extending each valuation to the whole space of the configurations. Shenoy and Shafer have shown that this schema can be applied if the combination and marginalization operators satisfy the following three axioms:

Axiom A1 (Commutativity and associativity of combination): Suppose G,H,K are valuations on g,h and k, respectively. Then

$$G \otimes H = H \otimes G \text{ and } G \otimes (H \otimes K) = (G \otimes H) \otimes K$$

[2] For the sake of simplicity we will not take into account in this paper "proper valuations", a subset of the valuations used to restrict the applicability the operators. Therefore, the definitions given here are not complete, but they preserve the basic ideas underlying valuation-based languages.

Axiom A2 (Consonance of marginalization): Suppose G is a valuation on g, and suppose $k \subseteq h \subseteq g$. Then

$$(G^{\downarrow h})^{\downarrow k} = G^{\downarrow k}.$$

Axiom A3 (Distributivity of marginalization over combination): Suppose G and H are valuations on g and h, respectively. Then

$$(G \otimes H)^{\downarrow g} = G \otimes (H^{\downarrow g \cap h})$$

2.2 Local Computation and Uncertainty Management

It will be useful to give now some examples of possible interpretations for the syntactical entities of a valuation-based language. This will show in which way a valuation-based language can be used for modelling different existing uncertainty theories. For probability theory, belief-functions, and a Boolean case, the mapping of the theory into the concepts of a valuation-based language have been proposed by Shenoy and Shafer (Shenoy and Shafer, 1988; Shenoy, 1989). For possibility theory, we use the mapping proposed by Dubois and Prade (Dubois and Prade, 1990). All these four interpretations satisfy axioms A1, A2 and A3 above. Thus, local computation can be used for them.

Probability:

Valuations on h are (unnormalized) probability distributions on the configurations of h

Combination: If G and H are probability distributions on g and h, respectively, then their combination is the probability distribution on $g \cup h$ defined by

$$(G \otimes H)(x) = G(x^{\downarrow g})H(x^{\downarrow h}) \quad \text{for all } x \in W_{g \cup h}.$$

Marginalization: If $h \subseteq g$ and G is a probability distribution on g, then the marginal of G for h is the probability distribution on h defined by[3]:

$$G^{\downarrow h}(x) = \Sigma\{G(x,y) \mid y \in W_{g-h}\} \quad \text{for all } x \in W_h$$

Belief Functions:

Valuations on h are basic probability assignment functions on sets of configurations of h.

[3] If g=h then $G^{\downarrow h}(x) = G(x)$. This applies also to the other interpretations given here.

Combination: If G and H are basic probability assignments on g and h, respectively, then their combination is the basic probability assignment on g∪h defined by[4]

$$(G\otimes H)(c) = \Sigma\{G(a)H(b) \mid (a^{\uparrow(g\cup h)})\cap(b^{\uparrow(g\cup h)})=c\} \text{ for all } c\subseteq W_{g\cup h},\ a\subseteq W_g, b\subseteq W_h$$

Marginalization: If $h\subseteq g$ and G is a basic probability assignment on g, then the marginal of G for h is the basic probability assignment on h defined by

$$G^{\downarrow h}(a) = \Sigma\{G(b) \mid b\subseteq W_g \text{ such that } b^{\downarrow h} = a\} \text{ for all } a\subseteq W_h.$$

Boolean:

Valuations on h are functions $H: W_h \rightarrow \{\text{true, false}\}$.

Combination: If G and H are valuations on g and h, respectively, then their combination is the valuation on g∪h defined, for all $x\in W_{g\cup h}$, by

$$(G\otimes H)(x) = \begin{cases} \text{true} & \text{if } G(x^{\downarrow g}) = \text{true and } H(x^{\downarrow h}) = \text{true} \\ \text{false} & \text{otherwise} \end{cases}$$

Marginalization: If $h\subseteq g$ and G is a valuation on g, then the marginal of G for h is the valuation on h defined, for all $x\in W_h$, by

$$G^{\downarrow h}(x) = \begin{cases} \text{true} & \text{if there is a } y\in W_{g-h} \text{ such that } G(x,y) = \text{true} \\ \text{false} & \text{otherwise} \end{cases}$$

Possibility:

Valuations on h are possibility distributions on sets of configurations of h.

Combination: If G and H are possibility distributions on g and h, respectively, then their combination is the possibility distribution on g∪h defined by

$$(G\otimes H)(x) = \min\,(G(x^{\downarrow g}), H(x^{\downarrow h})) \text{ for all } x\in W_{g\cup h}.$$

Marginalization: If $h\subseteq g$ and G is a possibility distribution on g, then the marginal of G for h is the possibility distribution on h defined by

$$G^{\downarrow h}(x) = \sup\{G(x,y) \mid y\in W_{g-h}\} \text{ for all } x\in W_h.$$

[4] This corresponds to usual Dempster's rule of combination (Dempster, 1966).

2.3. System's Overview

Pulcinella is a system for building and evaluating valuation systems based on Shenoy and Shafer's local computation technique. The system implements the general framework discussed above: it may be specialized to a given uncertainty theory by choosing an interpretation for the objects and the operators. Pulcinella is written in Lisp, and appears as a library of Lisp functions for creating, modifying and evaluating a valuation system. Alternatively, the user can choose to interact with Pulcinella via a graphical interface. The system builds on Hong Xu's implementation of the belief functions propagation technique of Shafer and Shenoy (Xu, 1991). The key move for generalizing Xu's program has been to parametrize it over a set of functions. At the moment, four interpretations of the general framework have been implemented: belief function, probability, Boolean and possibility (in the following, we will use the term "specialization" to refer to an implemented interpretation). By selecting one of these specializations, the user can transform Pulcinella in a probability propagation system, in a belief function propagation system, and so on. Moreover, Pulcinella is meant to be an open system: the set of functions which have to be defined to create a new specialization is very small and with a well defined semantics (reflecting the elements of a valuation-based language). Thus, it is easy for the user to define further specializations. We will first show how to use the existing specializations, and then discuss how a new specialization can be created.

As far as modelling quantitative knowledge is not concerned, the way to work with Pulcinella is the same for all specializations. The first thing the user has to do when he is face-to-face with Pulcinella, is to tell it which uncertainty theory he wants to use. Then, he can start to model his problem, either graphically or by calling Lisp functions. In terms of Shafer and Shenoy's framework, this means to create a valuation system representing his problem. This modelling process comprises two steps. In the first step, the user specifies the structural knowledge of his problem. This means defining all the variables to be used (along with their frames), and indicating which variables are linked together by a relation. By defining the variables and the relations the user implicitly fixes the subsets of variables for which valuations can be specified in the second step. The second step consists in modelling the quantitative knowledge: i.e. in defining the valuations on (some of) the subsets identified by the structural model created. This step depends from the specialization chosen. However, during this step, the user should keep in mind that a default valuation is given by the system to each variable and relation

if no valuation is defined by the user. Once the valuation system has been completely defined, the user can evaluate it by asking Pulcinella to start propagation.

Pulcinella is an open system. The user can build his own specialization with Pulcinella. To build a new specialization the first thing he has to do is to express his theory in terms of the syntactical entities of valuation-based languages, and to prove that the axioms of local computation are satisfied. Having expressed the theory in this way, he may now implement the functions specific to the new specialization. This is made easier by the clear semantics given to these functions: basically, they mirror the concepts and entities of a valuation-based language. These functions may be divided up into three groups:

1st GROUP	2nd GROUP	3rd GROUP
init-value	combine	normalize
init-joint-value	marginalize	propagate-end
init-configuration		

The functions of the first two groups correspond to the syntactical entities of the valuation-based language. The first group collects the functions for initializing the objects of the language: "init-value" must specify the default valuation of a single variable; "init-joint-value" specifies the default valuation of a set of variables; and "init-configuration" builds the set of configurations corresponding to a set of variables. The last function is not necessary from a theoretical point of view, but it allows the user to choose different data-structures for representing configurations in different specializations (e.g. choosing to privilege efficiency vs. simplicity). These data-structures may be chosen for each specialization independently, but have to obey some restrictions[5].

The second group comprises the two functions that embody the combination and marginalization operators. This is clearly the conceptual kernel of a specialization.

The third group includes extra functions for changing the valuations in "some way". One function must implement the normalization procedure for valuations. The second function is called after the propagation has been completed, and allows the builder of a specialization to perform some housekeeping before the results are shown to the user.

[5] The suspicious reader may be reassured: these restrictions will not force him to use terribly complicated data-structures.

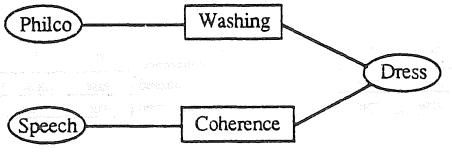
3. Case Study 1

3.1. The Problem

Our first example is adapted from (Saffiotti, 1987). We want to guess if Francesco will came wearing a black (B), white (W) or polka-dot (P) suite. We do not have any information about Francesco's preference (*state 0*), but we do know that his "Philco" washing machine is out; this makes us believe (say 80%) that he cannot choose W (*state 1*). Later, we remember that Francesco said yesterday that he dislikes mono-chromatic clothes; this is, for the notorious coherence of Francesco, a strong (say 90%) evidence both against B and W (*state 2*).

3.2. The Solutions

We first build a structural model for our problem. We define three variables: *Dress*, with frame {B, W, P}; *Philco*, with frame {ok, out}; and *Speech*, with frame {uttered, unuttered}. We then define two relations: *Washing*, between *Philco* and *Dress*; and *Coherence*, between *Speech* and *Dress*. The intended meaning of these elements should be self evident. The following is a graphical representation of our model, as appearing on the screen of Pulcinella:



The next step consists in deciding one uncertainty calculus to use, and to specialize Pulcinella to it. Suppose we choose to specialize Pulcinella to probability[6]. A dependency between variables is then encoded by a (unnormalized) joint probability distribution. The default valuation is the uniform probability distribution: if nothing is known about a variable or relation, all configurations are considered to have the same probability. We can then enter the distributions for our variables and relations. These distributions, which encode the quantitative knowledge in our problem, are shown below[7]:

[6] In practice, this reduces to selecting a menu item, or to evaluating the form "(specialize-uncertainty 'probability)".
[7] The valuations for these variables are obvious, and will not considered again in the next examples.

possible, and is expressed by a mapping from configurations to the interval [0,1]. The default possibility distributions is given by attaching to each configuration the value 1: if nothing is known about a set of variables, all configurations are regarded as completely possible. After switching Pulcinella to the possibilistic interpretation, we enter two possibility distributions for our relations. These distributions, and the one obtained for the variable *Dress* after propagation, are shown below.

$\Pi_{washing}$

	B	W	P
ok	1	1	1
out	1	0.2	1

$\Pi_{coherence}$

	B	W	P
uttered	0.1	0.1	1
unuttered	1	1	1

Value	*State 0*	*State 1*	*State 2*
B	1	1	0.1
W	1	0.2	0.1
P	1	1	1

Finally, we consider the case in which we collapse uncertainty to true/false values. These values are better understood in term of satisfaction of constraints: a value true for a configuration means that this configuration is acceptable given the constraints that characterize our problem. Accordingly, a relation among variables will be encoded by selecting all those configurations that are admissible, and by associating true to them. The default valuation is given by attaching to each configuration the value true: if nothing is known about a set of variables, each configuration can be true. The following tables show the values for our relations, and the results obtained, with the Boolean specialization.

$\mathcal{T}_{washing}$

	B	W	P
ok	true	true	true
out	true	false	true

$\mathcal{T}_{coherence}$

	B	W	P
uttered	false	false	true
unuttered	true	true	true

Value	*State 0*	*State 1*	*State 2*
B	true	true	false
W	true	false	false
P	true	true	true

3.3. Discussion

In this example, as in the next one, we can recognize three basic categories of differences:

1) differences in the data we have to provide (which data, how many, in which form, . . .);

2) differences in the results due to differences in the data we provided; and

$\mathcal{P}_{washing}$

	B	W	P
ok	1/3	1/3	1/3
out	0.4	0.2	0.4

$\mathcal{P}_{coherence}$

	B	W	P
uttered	0.05	0.05	0.9
unuttered	1/3	1/3	1/3

$\mathcal{P}_{Philco}$

ok	0
out	1

$\mathcal{P}_{Speech}$

uttered	1
unuttered	0

Finally, we ask Pulcinella to start propagation. The following table gives the marginal probabilities computed for *Dress* at different moments[8]:

Value	*State 0*	*State 1*	*State 2*
B	0.33	0.4	0.05
W	0.33	0.2	0.03
P	0.33	0.4	0.92

Suppose that we now want to try to use belief functions for modelling our problem. All we have to do, is to specialize Pulcinella to belief functions, and to input the quantitative knowledge of the problem in the form of two basic probability assignments representing our relations:

Subset	$m_{washing}$
ok {B, W, P} out ∅	0.8

Subset	$m_{coherence}$
uttered {B, W, P} unuttered ∅	0.9

Intuitively, the subset to which $m_{washing}$ assigns a 0.8 mass represents the fact that the answer to our problem may be any of B, W and P when *Philco* = ok, and any of B and P when *Philco* = out. The remaining 0.2 mass is automatically given to the whole frame. The default valuation is the basic probability assignment function which attach the value 1 to the whole set of configurations. After propagation, we get the following results for the variable *Dress*[9]

	State 0		*State 1*		*State 2*	
Value	bel	pl	bel	pl	bel	pl
B	0	1	0	1	0	0.1
W	0	1	0	0.2	0	0.02
P	0	1	0	1	0.9	1

Next, we consider using possibility theory. In this specialization, the user models quantitative knowledge by specifying possibility distributions on the defined sets of variables. A possibility distribution on a set of variables reflects to what extend each configuration of the set of variables is

[8] States in the table refer to the states mentioned in the statement of our story.

[9] For greater readibility, we show the results using *bel* and *pl* functions (Shafer, 1976).

3) differences in the results due to differences in the mechanisms used in the theories.

Concerning category 1 above, the quantitative knowledge stated in the problem has to be coded differently in the four specializations. In the probabilistic case, the need to specify completely the joint distributions $\mathcal{P}_{washing}$ and $\mathcal{P}_{coherent}$ has obliged us to replace in some way the missing information. In particular, in $\mathcal{P}_{washing}$, the 80% probability of (B$\vee$W) has been converted into exact probability values for B and P individually (arbitrary assuming equiprobability); a similar operation has been made for $\mathcal{P}_{coherent}$. On the other hand, in both the belief function and the possibilistic specializations knowledge has been expressed at exactly the level of granularity that is available. It must be noticed that using basic probability assignments to encode knowledge in the belief function specialization may sometimes be less intuitive (mainly because we have to work on subsets). Finally, our knowledge has easily been reduced in the obvious way in the Boolean specialization.

Moving now to the analysis of the results (categories 2 and 3 above), we first notice that —as expected— all four specializations agree from the qualitative viewpoint. The major quantitative differences show up between the results of the probabilistic specialization and those of the belief function (or possibilistic) one. These differences must be tracked back to differences in the data used, mainly because of the additional hypotheses (equiprobability) introduced in the probabilistic case. Remarkably, these differences are particularly evident in those cases (States 0 and 1) in which ignorance is predominant: here, in spite of our ignorance, the probabilistic approach needs (and gives) precise figures[10]. Another interesting difference arises between the belief function and the possibilistic cases. Here, differences in results do not originate from differences in the inputs given, but rather from the different combination mechanisms. To wit, consider State 2: the Philco evidence and the Speech evidence are both denying the W hypothesis. These two items of evidence have been combined in one much stronger evidence against W by Dempster's rule in the belief function case, resulting in a very small (0.02) plausibility for W. On the other hand, the stronger of them has simply been selected through the MIN operator in the possibilistic approach, giving a possibility value of 0.1 for W.

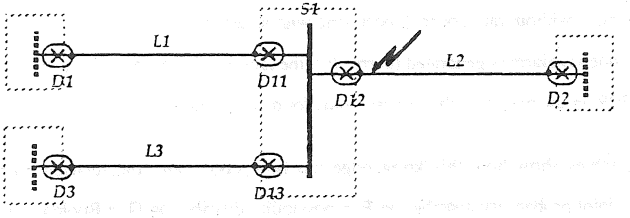
10 «Wovon man nicht sprechen kann, darüber muß man schweigen» (What we cannot speak about we must consign to silence) (Wittgenstein, 1921).

4 Case Study 2

The above considerations illustrates how to use Pulcinella for analyzing the different behaviours of uncertainty treating theories. However, we do not have up to now a pragmatic unit for measuring how much do the different theories fit our needs. Still, Pulcinella may be also used as a tool for evaluating uncertainty formalisms in the light of the uses to which they are to be put. To this respect, we now present our second case.

4.1. The Problem

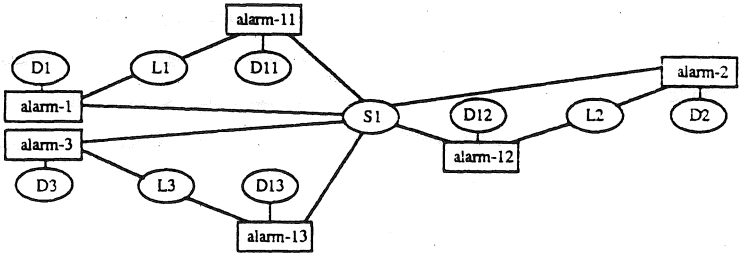
This problem has been tailored on an experiment made in modelling the uncertainty present in a problem of fault diagnoses in electricity networks. For the sake of clarity, both the qualitative and the quantitative knowledge have been greatly simplified. The full experiment, and the actual figures used, are reported in (Saffiotti and Umkehrer, 1991b). We consider here the following fragment of an electricity network:



This fragment comprises four substations, linked by three lines L1, L2 and L3. The substation in the middle includes S1, a big conductive bar used for connecting more lines together. The Di's are "circuit breakers": devices that can separate two lines, and that watch the part of the network on their "hot" side (marked by a dot in the picture) for overloads. When an overload is detected, a circuit breaker may generate two kinds of alarms: "instantaneous", for "big" overloads (normally caused by a fault in the line the device is on); or "delayed", for "small" overloads (normally caused by a fault in a neighbour line).

4.2. The Solutions

We can model the above description in Pulcinella by the following variables and relations:



where *Di*'s are variables representing circuit breaker states, with possible values ok (no alarm), del (delayed alarm), and inst (instantaneous alarm); *Li*'s and *S1* represent line states, with frame {ok, fault}; and the *alarm-i*'s relate generation of alarms by breakers with states of neighbour lines.

The quantitative knowledge given by the experts is not very rich. Essentially, it says that:

1. alarms are not very reliable: in roughly 10% of the cases, they do not correspond to the real situation (alarm generated without fault, or fault occurring without alarm);
2. if an instantaneous alarm is generated (correctly), the fault is 70% of the cases in the line the breaker is on, and 30% in the next one; the reverse holds for delayed alarms.

The following tables show how this knowledge has been coded into the relation *alarm-1* with an (unnormalized) joint probability distribution $\mathcal{P}$, a possibility distribution Π, a Boolean constraint $\mathcal{T}$, or a basic probability assignment m, respectively. Distributions for the other *alarm-i*'s are similar; however, those for the *alarm-ij*'s are different.

$\mathcal{P}(\cdot)$

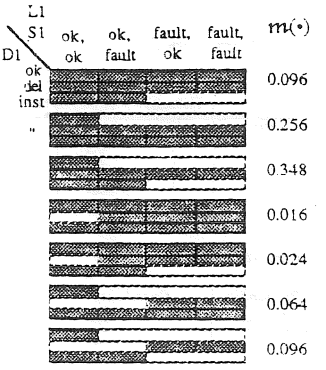
L1, S1	D1 ok	D1 del	D1 inst
ok, ok	0.9	0.1	0.1
ok, fault	0.05	0.6	0.2
fault, ok	0.05	0.2	0.6
fault, fault	0.001	0.1	0.1

$\Pi(\cdot)$

	ok	del	inst
ok, ok	1	0.1	0.1
ok, fault	0.1	0.7	0.3
fault, ok	0.1	0.3	0.7
fault, fault	0.1	1	1

$\mathcal{T}(\cdot)$

	ok	del	inst
ok, ok	true	false	false
ok, fault	false	true	true
fault, ok	false	true	true
fault, fault	false	true	true



We now imagine a scenario in which a fault occurs on line L2 very near to D12 (see the above picture); consequently, a "delayed" alarm is sent by D2, and a "instantaneous" alarm by D12. Moreover, we imagine that D1 has been adjusted incorrectly (it happens), and is too sensitive: thus, also D1 sends a "delayed" alarm. Three main hypotheses are compatible with this set of alarms, shown in order of preference:

1) Fault in L2; alarm from D1 spurious;
2) Fault in S1; missing alarm from D3, and alarm from D12 spurious;
3) Fault in L1; D11 not working (missing alarm & did not open), missing alarm from D3, and alarm from D12 spurious.

The following tables summarize the results obtained by running Pulcinella on this example with the probability, possibility, Boolean, and belief function specializations, respectively.

1. After receiving the alarm from D2

	Probability	*Possibility*		*Belief Functions*		*Boolean*	
Var	$\mathcal{P}$(fault)	Π(fault)	Π(ok)	bel(fault)	bel(ok)	$\mathcal{T}$(fault)	$\mathcal{T}$(ok)
L1	0.002	1	0.7	0.02	0	true	true
L2	0.226	1	0.7	0.02	0	true	true
L3	0.002	1	0.7	0.02	0	true	true
S1	0.087	1	0.3	0.16	0	true	true

2. After receiving the alarm from D12

	Probability	*Possibility*		*Belief Functions*		*Boolean*	
Var	$\mathcal{P}$(fault)	Π(fault)	Π(ok)	bel(fault)	bel(ok)	$\mathcal{T}$(fault)	$\mathcal{T}$(ok)
L1	0.002	1	0.7	0.02	0	true	true
L2	0.972	1	0.1	0.60	0	true	false
L3	0.002	1	0.7	0.02	0	true	true
S1	0.013	1	0.3	0.16	0	true	true

3. After receiving the alarm from D1

	Probability	*Possibility*		*Belief Functions*		*Boolean*	
Var	$\mathcal{P}$(fault)	Π(fault)	Π(ok)	bel(fault)	bel(ok)	$\mathcal{T}$(fault)	$\mathcal{T}$(ok)
L1	0.219	1	0.7	0.03	0	true	true
L2	0.919	1	0.1	0.60	0	true	false
L3	0.002	1	0.7	0.03	0	true	true
S1	0.141	1	0.3	0.29	0	true	true

4.3. Discussion

As in Section 3, we first consider the differences in the inputs required by the different techniques (category 1 above), and than the differences in the output produced (categories 2 and 3). However, we try to take here a more "knowledge engineering" viewpoint. Probability theory requires several values that are not available, or not directly available, in our data (e.g. $\mathcal{P}$({ok, ok, fault} and $\mathcal{P}$({ok, fault, fault}): these values had to be computed from the available information, by introducing hypotheses of equiprobability or independence. Notice that the amount of subjective estimates required by the probabilistic approach greatly increases the risk of inconsistency among them (but see Duda et al., 1986). On the contrary, the process of supplying data is extremely simple in the possibility and the belief function specializations: the expert must supply just the data she knows, and consign the rest to silence. We do not need to force her (or the figures she supplies) to say something they were not meant to. However, in the belief function case, the translation of this data into basic probability assignments may sometimes be hard. The difficulty of the Boolean case is somewhat complementary to that of the probabilistic case: the data given by the expert has to be rounded very roughly, and she may feel uncomfortable with the approximations obtained. A possible reaction to this rude attitude is to try to split general rules into more specific ones. The aim of this would be to reduce uncertainty by explicitly accounting for the possible exceptions to rules. Though this constitutes a stimulus for the expert that may sometimes result in a better explication of her knowledge, the strongly empirical, tangled and

"artistic" nature of electrical fault diagnostic knowledge discouraged us from using the Boolean specialization for our problem.

If we now switch to considering the results of our example, we can observe that our four specializations have produced results that are both quantitatively and qualitatively different. The first phenomenon we notice is that the Boolean specialization does not suggest the possibility of a fault in S1. The causes of this reside both in the input given and in the combination mechanism used. As for the input, the knowledge encoded in our relations allows us to infer that the fault is, e.g., either in S1 or in L1. Yet, no relation encodes knowledge which allows us to infer the S1 hypothesis alone. Discrimination between S1 and L1 is, on the other side, captured by differences in the given weights in the other approaches. As for the mechanism, aggregating knowledge by an AND operation does not allow to perform that "counting" of evidence that seems necessary if we want to accumulate items of weak evidence together. In our example, the evidence given by D2 only partly supports the hypothesis S1; however, we expect that further support to S1 coming from D1 should reinforce the S1 hypothesis. One way to fix this problem in the Boolean specialization is to add new rules (e.g. "IF at least 2 among D1, D2 and D3 send a delayed alarm, THEN S1 is faulty"); but the lesson taken from this story seems to be that considering uncertainty in our problem is necessary, if we want to preserve inferential power without having to drown in a see of specialized rules.

The second qualitative difference we want to notice regards the hypothesis "*L1* = fault", which is suggested by the possibilistic and the probabilistic[11] specializations, but not by the belief function one. The origin of this is again in the way we have coded our data: a delayed alarm does not support, in the belief function case, a fault in the adjacent line *individually*, but a fault in the adjacent *or* in the next lines[12]. On the contrary, in the possibilistic and the probabilistic cases, the evidence given by the alarm has to be spread among each hypothesis individually. A related difference concerns the hypothesis "*L3* = fault", which is suggested by the possibilistic specialization only. This "over-inferencing" is rooted in the fact that we have taken a rather strong attitude when defining the possibility distributions for the *alarm-i*'s relations (viz. the frequential knowledge about the localization of the fault has been converted

11 Notice that the possibilistic specialization supports this hypothesis by partially denying that "*L1* = ok".

12 But notice that, differently from the Boolean case, further evidence may convert this support to a support for the adjacent line individually.

to a measure of (im)possibility). A less committed interpretation of our data would lead to the following distribution

$\Pi(\cdot)$	ok	del	inst
ok, ok	1	0.1	0.1
ok, fault	0.1	1	1
fault, ok	0.1	1	1
fault, fault	0.1	1	1

where the information about the relative support given by an alarm to a near fault or a far fault (which is not, strictly speaking, a matter of possibility) has been ignored. Using this encoding, we would get a result suggesting L2 alone (like in the Boolean case). The lesson to be taken here seems to be that possibility theory may provide the additional inference power that we need in our problem, but this power cannot be "fine-tuned" easily.

As for the quantitative differences, the most important one (which might be regarded as qualitative as well) concerns the hypothesis "S1 = fault". This hypothesis is reinforced by the arrival of the alarm from D1 in the probabilistic and belief function specializations, but not in the possibilistic one. The cause here is only the combination mechanism used: like the AND of the Boolean case, the MIN operator does not allow us to perform that "counting", which seems necessary in our problem to accumulate evidence correctly. As a consequence, the answers given by the possibilistic specialization to our problem are mainly meaningful from the qualitative viewpoint (they allow us to focus on those hypotheses which are possible), but they are fairly poor from the quantitative one. To this respect, we notice that the results given by the probabilistic specialization are very rich from the quantitative viewpoint; though, because of our straining the input data, this precision may be somehow unjustified. The belief function specialization seems to provide the tradeoff between precision and non-commitment that best fits our knowledge. Yet, the computational complexity inherent in belief functions shows up in running the full experiment: the final choice of the uncertainty treating technique to be used for solving the full scale diagnostic problem will have to take this factor into consideration.

5. Conclusions

Whenever we have to deal with a problem involving uncertain knowledge, choosing the uncertainty management technique which best fits it becomes of primary importance. To this respect, the approach we have proposed is peculiar in its being experimentally grounded. There are at least two

places where such an approach, and the tool it is based on, is expected to be useful. First, in the room of the theorist involved in the comparative study of uncertain reasoning models. For her, Pulcinella might play the role of a slide-rule in her testing the behaviour of uncertainty theories over sample problems of academic interest. The fact that the structural model of the problem is the same for all theories ensures the soundness of the test. Second, in the office of the knowledge engineer. Here, Pulcinella might prove helpful in checking out different uncertainty management techniques for solving the problem at hands, and then judging—on this experimental basis—which one appears to be the most adequate to our case. In particular, we may test both the input requirements and the obtained results of each theory, and evaluate how the available data is accommodated for, or our expectations satisfied.

Pulcinella belongs to the growing community of systems for representing and propagating uncertainty in networks (e.g. Andersen et al., 1989; Hsia and Shenoy, 1989; Zarley et al., 1988; Xu, 1991). However, to our knowledge, it is the only existing system that can propagate uncertainty values according to more than one uncertainty theory.

Acknowledgements

The present research has benefit from discussions with (and comments from) Fabrizio Sebastiani, Yen-Teh Hsia, Robert Kennes, Bruno Marchal, Philippe Smets and Xu Hong.

References

Andersen, S.K., Olesen, K.G., Jensen, F.V. and Jensen, F. (1989) "HUGIN - a Shell for Building Bayesian Belief Universes for Expert Systems", *Procs. of the 11th Int. Congress on AI* (Detroit).

Chatalic, P., Dubois, D. and Prade, H. (1987) "A System for Handling Relational Dependencies in Approximate Reasoning", *Procs. of the 3rd Int. Expert System Conf.* (London, GB) 495-502.

Cheeseman, P. (1985) "In Defense of Probability", *Procs. of IJCAI-85*: 1002-1009.

Clark, D. (1988) (Ed.) "Responses to 'An AI view of the treatment of uncertainty' by Alessandro Saffiotti", *The Knowledge engineering Review* 3(1) 59-91.

Dempster, A.P. (1966) "Upper and Lower Probabilities Induced by a Multivalued Mapping", *Annals of Mathematical Statistic* 38: 325-339.

Dubois, D. and Prade, H. (1990) "Inference in possibilistic hypergraphs", Proc. of IPMU Conference.

Duda, R.O., Hart, P.E. and Nilsson, N.J. (1986) "Subjective Bayesian Methods for Rule-Based Inference Systems", *AFIPS Procs.* 45 (New York) 1075-1082.

Fox, J. (1986) "Three Arguments for Extending the Framework of Probability", in Kanal, L.N. and Lemmer, J.F. (Eds.) *Uncertainty in AI* (North Holland).

Hsia, Y. and Shenoy, P.P. (1989) "MacEvidence: A Visual Environment for Constructing and Evaluating Evidential Systems", *Working Paper No. 211* (School of Business, University of Kansas, Lawrence).

Lauritzen, S.L. and Spiegelhalter, D.J. (1988) "Local Computations with Probabilities on Graphical Structures and Their Applications to Expert Systems", *Journal of the Royal Statistical Society B* **50**(2) 157-224.

Pearl, J. (1988) *Probabilistic Reasoning in Intelligent Systems* (Morgan Kaufman, CA).

Saffiotti, A. (1987) "An AI view of the treatment of uncertainty", *The Knowledge Engineering Review* 2(2) (Cambridge University Press, UK) 75-97.

Saffiotti, A. (1989) "De Propagationibus", *ARCHON Technical Note IRIDIA/ARCHON/TN-05*, (IRIDIA, Université Libre de Bruxelles, Bruxelles).

Saffiotti, A. and Umkehrer, E. (1991a) "PULCINELLA User's Manual", *Technical Report IRIDIA/91-5*, (IRIDIA, Université Libre de Bruxelles, Bruxelles).

Saffiotti, A. and Umkehrer, E. (1991b) "What do you want to propagate for breakfast ?", *Technical Report IRIDIA/91-7*, (IRIDIA, Université Libre de Bruxelles, Bruxelles).

Shafer, G. (1976) *A Mathematical Theory of Evidence* (Princeton University Press, Princeton).

Shafer, G., Shenoy, P.P. and Mellouli, K. (1987) "Propagating Belief Functions in Qualitative Markov Trees", *International Journal of Approximate Reasoning* 1: 349-400.

Shenoy, P.P. (1989) "A Valuation-based Language for Expert Systems", *International Journal of Approximate Reasoning* 3: 383-411.

Shenoy, P.P. and Shafer, G. (1988) "An Axiomatic Framework for Bayesian and Belief-Function Propagation", Proc. of AAAI Workshop on Uncertainty in AI: 307-314.

Smets, Ph. (1988) "Belief Functions", in: Smets, Ph., Mamdani, E.H., Dubois, D. and Prade, H. (Eds.) *Non-Standard Logics for Automated Reasoning* (Academic Press, London).

Szolovits, P. and Pauker, S.G. (1978) "Categorical and Probabilistic Reasoning in Medical Diagnosis", *Artificial Intelligence* 11: 115-144.

Wittgenstein, L. (1921) *Tractatus Logico-Philosophicus* (Annalen der Naturphilosophie).

Xu, H. (1991) "An Efficient Implementation of Belief Function Propagation", *Technical Report IRIDIA/91-4*, (IRIDIA, Université Libre de Bruxelles, Bruxelles).

Zarley, D.K., Hsia, Y. and Shafer, G. R. (1988) "Evidential Reasoning using DELIEF", *Working Paper No. 193* (School of Business, University of Kansas, Lawrence).

DEFAULT REASONING AS A DECISION PROCESS ON INDUCTIVE INFERENCES

Fernando Tohmé[1] and Claudio Delrieux[2]
Universidad Nacional del Sur
Alem 1253
Bahía Blanca, Argentina

[1] Departamento de Economía
[2] Departamento de Ingeniería Electricista

ABSTRACT

Default reasoning has acquired importance in AI as a way to represent knowledge and common sense reasoning. After early fixed-point-based propositional entailment methods declined, classical approaches in the neat quarters has been to achieve nonmonotonicity in a syntactical manner. Dissent, however, has emphasized a decision theoretic point of view from which certain, perhaps limiting, results can be derived.

This paper follows the last approach, showing how Social Choice theory provides a unified framework where default reasoning methods can be embedded. Besides, in a topological analysis of the structure of a default theory, it will be shown that there exists two strata in such a theory, a deductive one and a default one.

The links of this approach with current conceptions about the nature of scientific theories is discussed in relation to possible implementations of the presented method.

I- Introduction:

One of the main goals of AI is to mechanize the reasoning process, and particulary the cognitive feature called *common sense* This goal is far from being attained, due to the lack of well suited formal methods to manage it. Historicaly this quest belong to Logic. But the limitative theorems proved in the 30´s by Gödel, Church and Turing convinced many that the philosophical rôle of Logic should be the study of methods able to verify the correctness of reasoning, that is, that of an a posteriori analytic.

Such a revision of the rôle of Logic in AI appeared as a critique of predicate calculus as a representation of common sense reasoning. The most criticized feature of formalized theories in first order languages was **monotonicity**. Therefore, a great deal of effort was dedicated to the research of nonmonotonic methods following a clear tendency: to modify FOL to handle nonmonotonicity inside the formalism [1]. This approach generated a number of different formalisms as **circumscription [9], default logic [15], nonmonotonic logic [10]** (and its embellished version, **autoepistemic logic [11]**), **the logics of change and action [18], the semantics of inheritance networks [19]**, etc. To check conclusions, all these formalisms have to verify consistency of theories. This means that new inconvenients as the non validity of the classical metatheorems (deduction, consistency, etc.) are added to the traditional formal problems.

Many authors proposed alternative approaches, noteworthy the consideration of revision of theories not as a logical problem but as an epistemological one [6], and the rational handling of inferences (in a decision theoretic setting) [2]. One of the pourposes of this paper is to show how and why these heterodox criteria can be embedded in a game theoretical frame. The crucial step, in this direction is to show the evolution of theories as a **conclusion-choice process**. Theories stated in a logical language can be seen as "moves" in a game against *Reality* (the evidential context of theories). Added evidence to a theory is then "a move of Reality", followed by a reformulation of the theory as an answer.

With tools borrowed from the theory or Social Choice (a kind of decision theory with multiple criteria) it will be shown how it is possible and indeed more perspicuous to determine methods to mechanize the beforesaid game, and the effect of these methods on certain topological features of theories will be analyzed.

In section II, the structure of reasoning systems that handle default theories will be studied from an epistemological point of view. In section III an overview of the main features of Social Choice Theory will be given, related to the interpretation of default theories. Next section will be concerned with the topological structure of default theories as a help in the design of more robust methods. In section V a couple of examples are presented and in the last section the general conclusions and forthcoming work is discussed.

II - Default reasoning from an epistemological point of view:

In spite of a common structure, scientific theories are of a different kind that the strictly logical ones. There are two proper levels in every theory: the syntactical and the semantical ones, being the first the form of expression and the second the expressed concept. The first difference appears in the semantical aspect, because models in a scientific theory should be limited in number. This is due to the fact that the meaning of a theory of this kind depends not only from its syntactical structure (as in purely logical theories) but also, and mainly, from a third semiotic level: pragmatics (the relation of the theory with its users and with Reality).

The design of theories about any fragment of Reality has to take into account the three mentioned aspects, but recognizing syntax as (obviously) subordinated to semantics and pragmatics. This implies that if contextual information of the model is modified, the pragmatical relations between context and model will force a syntactical modification of the theory, consisting in a reformulation of its statements or in a relaxation of the grammatical constraints on its language.

A fundamental epistemological problem has been to define clearly how and when the syntax of a scientific theory has to be modified. A first proposal was given by K. Popper, who considered that the only syntactical feasible inferences in a theory are deductive (meaning that all the information needed to draw a conclusion has to be in its antecedents). Therefore, the negative contrastation of a theory has to be attributed to an error, forcing the withdraw of the theory and the design of a better one. That is to mantain the grammar and write new statements [14].

However, T. Kuhn, examining historical examples concluded that scientists faced to negative evidence reformulate their theories partially, and only when it is absolutelly impossible to follow this procedure, just then a new theory is designed [7]. This

approach was followed by many authors adscribed to the so called *structuralist* school. They proposed formalizations for this process, characterized by distinguishing several strata of stability in a theory. The most stable stratum is the hard core of the theory, which in case of not being correct forces the failure of all the theory. Within this conception, a theory posseses several models corresponding to all feasible modifications. Any theory in this view, shows no equality between its **intensional** and **extensional** contents, so being **nonmonotonic**.

Despite its conceptual clarity, postkuhnian epistemology is still susceptible to the remark that considering only **deductive inferences** obscures the truly tentative nature of a scientific theory. Besides, while the popperian dynamic of theories obeys clearly the "all or nothing" feature of deductive logic, the postkuhnian script appears as a set of *ad hoc* rules without a deeper theorical justification.

Several proposals in AI for formalizing default reasoning are analogous in nature to postkuhnian epistemology, due to the obsesive fixation to deductive inference (perhaps because it is the most neatly formalized reasoning method available). However a slight evaluation of reasoning forms shows that besides the deductive one exist the inductive, the analogical, and the abductive or conjectural ones, being the last two the most important for scientifical thought and common sense. All the forms of reasoning, except deduction, are characterized by allowing to draw conclusions without enough information. But between them are remarkable differences, the main one being that only inductive inference can be stated through rules in which consequences are drawn tentatively from antecedents. Therefore, in case of common sense reasoning, inference rules can be considered **default rules**.

Inductive reasoning was always associated to probability theory [14], but in case of being considered as default rules acquires some distinctive features. Default rules were taxonomized in the following classes: **prototypical** (involving generic plural statements), **individual** (concerning normal states of objects, facts or situations) and **systemic** (concerning states of information relevant for a set of default rules) [2]. The crucial point here lies in that although the **vocabulary** of default theories is classical, their **grammar** is not so, and therefore they can be "inconsistent" in a deductive sense. Then, actual modification of a default theory occurs in its model. An unsuitability of this approach is that, lacking methods for handling conjectural reasoning (able to allow the design of new default rules up from a set of evidences), the only permissible procedure is to shorten the range of rules through the explicitation of exceptions. This is, of course, a failure common to all known default reasoning formalisms.

Given the computational frame of AI, a consideration on the implementation of a methodology to handle the evolution of default theories is in order. The point is to distinguish the four levels that should appear in any implementation:

a) Default rules (perhaps together with classical ones).
b) A database of facts and tentative conclusions.
c) A source of new evidence.
d) A mechanism of revision of theories.

It is worth noting that alterations required in d) are over the set defined in a), even though the direction almost pursued in current research is to modify the set defined in b). It should be clear that the most complicated part in the design of an implementation is the definition of the mechanism of revision of theories. This task requires an epistemological approach to provide a well grounded method.

III - Reconstruction of theories as a decision-theoretic process:

Are we constrained to attack the problems above only from the syntactical or the semantical side? Our intended answer is that pragmatic stances are more perspicuous and abarcative (the higher the better). In the proposed framework every default rule determines a bias in the model preference, which can be in turn restated by new incorporated evidence. Doyle's proposal is to see default rules as rational preference indicators [2], each one being a decision-theoretic criterion between alternative conclusions of a theory. This is enterely in accordance with the conception of default rules as inductive statements interpreted like rational decision rules [8] in the frame of subjective probability [14].

Contradictions between conclusions and evidences indicates the emergence of new decision criteria, because new evidence can be seen as default rules without antecedents. The conflict between criteria forces to choose one of them to mantain consistency in the set of conclusions.

This situation is similar to the votation problem: candidates are the conclusions of the theory and default rules the voters. Determination of the set of chosen conclusions is then the main problem for the modification of a default theory. This apparently trivial problem is a fundamental one from an epistemological standpoint. Inductive (and

default) theories are inconsistent if considered deductive. Therefore, models involving tentative conclusions cannot be obtained but with a refereeing between conlicting defaults. Features of this decision process determine the kind of default theories to be obtained, raising a multiplicity of possible models for the same set of proper rules (like scientific theories in the postkuhnian view). Votation is the paradigmatic problem of Social Choice theory. As it name indicates, Social Choice theory tries to determine the different ways in which the wills of several individuals in social aggregates can be combined to give place to a "social will".

In Social Choice theory, given a set of individuals G, it is required that each individual g should have a preferential ordering R(g) of the countable set of social states X in such a way that from the individual orderings a unique ordering R can be obtained for all G.

The problem is thus to determine R, given the individual R(g). K. Arrow proved that the following conditions (for a democratical determination of R) are incompatible, where P and I are relations of strict preference and of indiference, respectively, and apostrophes indicate alternative orderings:

1) *Positive responsiveness*

For any x,y in X, if for all g, xP(g)y--->xP'(g)y
and xI(g)y--->xR'(g)y
and for some g xI(g)y and xP'(g)y
then xRy--->xP'y

2) *Pareto principle*

if xR(g)y for all g then xRy

3) *Non dictatorship*

It doesn't exist g such that for all x,y
if xR(g)y then xRy

4) *Independence of irrelevant alternatives*

If for all g exist R(g) and R'(g) and for all
x, y in X, xR(g)y<--->xR'(g)y then R = R'

5) *Transitivity*

for all x,y,z in X if xRy and yRz then xRz

As it will be shown, incompatibility between these constraints is very important for the determination of the set of conclusions of a default theory, given the interpretation of X as the set of conclusions sets of the theory G, each g as a default rule and R(g) an ordering of sets of conclusions according to g.

Doyle includes an additional constraint, the principle of Paretian Optimality:

if xRy and yR(g)x for some g then exists g' such that xR(g')y

and proves that the stability of default theories (the existence of closed and complete extensions of them [14]) implies the Paretian optimality [2]. It is, however, enough to consider only the classical constraints 1 to 5, because Paretian optimality can be derived in a relaxed version of them, as it will be shown.

Incompatibility between constraints 1 to 5 is usually interpreted as indicating that a dictator has to exist (Arrow's theorem is also called the *dictator theorem*). To avoid this result condition 5 can be relaxed to provide the following constraint:

5') *Quasitransitivity*:

R is quasitransitive iff for x,y,z in X, xPy and yPz imply xPz

In this case it is proved that a relation R exist, satisfying 1 to 5' [4]. This relation can be characterized as follows:

xPy iff for all g xR(g)y and there exists g' such that xP(g')y

This definition is, in fact, the classical statement of the principle of Paretian optimality [5] and therefore (assuming the equivalence of stability of theories and Paretian

optimality, as proved by Doyle) that it is capable, when applied to default theories, to find stable sets of conclusions.

The proposed method of default theories interpretation is abarcative: if as an alternative method is considered one involving only rules 1 to 4, obeying Pareto principle, equivalent results to **circumscription, skeptical inheritance**, or CWA will be provided [2]. As the general method obeys also the Pareto principle, it is clear that these conservative forms of non default reasoning can be embedded in the forms handling stable theories. The only form of default reasoning not included, considering constraints 1 to 5' is the represented by **production rules**, (applied *sequentially*). This is of course equivalent to a principle of sequencial dictatorship.

Implementation of this method requires every default rule to have a preferential ordering on the subsets of conclusions. This order can be obtained simply by set inclusion, considering the inclusion of grounded conclusions of the rule in the subsets. It should be noted that the only rules defining a strict ordering are the classical ones, because default rules are purely tentative. Finally, taking the global order for the theory, the maximal set in this order is the new conclusions set of the theory, that is **the new model**. It is clear that although no formal complexity analysis has been performed, given the exponential growing of the power set of the set of conclusions due to the inclusion of new evidence, this method has NP complexity. Topology of ordered sets comes to the rescue, as it is shown in the next section.

IV - Topology of default theories.

In the last section was shown how, with a reduced number of conditions it is possible to establish a method for the reformulation of default theories in face of new evidence. It is clear that evidence will shorten the range of default rules, and unless a method to create new rules taking into account the limitations impossed by the growing information (a "fix" in the sense of Perlis [13]) is added, only the axioms and the conclusions derived through classical rules will remain in the conclusions set.

In KR, as in epistemology, fixes are undesiderable because they are scruffy, *ad hoc* and procedural, and rules continuosly incorporating exceptions in their body prevent better reformulations of the theory. The same can be said about default theories, which can be seen as sets of preferences over tentative conclusions. The fixes, providing immunity against errors (through a very poor kind of learning) are in this

context also condemnable. Thus, it is appropiate to analyze the form in which the body of rules contributes to the collapse of the theory, providing a taxonomy of stability strata, instead of trying to repair the damages caused by contradictions.

The key to determine this strata of stability in default theories is to analyze the topological structure determined by the proposed methods of reasoning, equivalent to well known default reasoning formalisms. The tools are again provided by Social Choice Theory.

An important notion, that allows to examine in detail the action of default rules as indicators of preference on subsets of conclusions, is that of decisive group [17] defined over a pair of alternatives x, y of X as a set of rules D(x,y) such that:

xRy iff xR(g)y for all g in D(x,y)

Evaluation of the topological structure defined by the methods of determination of R will be carried on the class O of decisive groups of rules to find out which subsets of rules will be more significative for the model of the theory when new evidence is incorporated.

To acomplish this goal lets consider some well known results on the topological properties of decisive sets. The notions of **filter**, **prefilter** and **ultrafilter** are the core of this topological analysis. Given two sets T and O, being the later a subset of the power set of the former that obeys the following constraints:

a- T pertains to O
b- If a set G pertains to O, and G is included in a set J, then J pertains to O
c- If the sets G_i are included in O for i=1..n then the intersection of the G_i is not empty
d- If G and J pertain to O, then their intersection pertains to O
e- If G doesn't pertain to O, its complement T-G does

O is a **prefilter** if satisfies a, b and c. If O satisfies also d it is called a **filter** and if it verifies all the properties it is an **ultrafilter**. It has been proved that:

- The Pareto principle (the method of determination of R satisfies 1 to 4) implies that O is a prefilter.

- Pareto optimality (the method satisfies 1 to 5') implies that O is a filter.

- Dictatorship (the method verifies 1 to 5) implies that O is an ultrafilter.

The first result can be interpreted as stating that there are a subset of T, not pertaining to O, such that each one of its elements pertains to each set of O, belonging to a *Collegium* whose elements have decision power over every pair of possible alternatives. In the case of Paretian optimality the second result states that the Collegium becomes a decisive set (an Oligarchy). The last result means simply that the oligarchy has only one element.

Relating these results with the stated in the last section, it is obvious that if T is the set of rules of a default theory (including classical rules) the more conservative methods of default reasoning will be characterized by having a Collegium, that is a set of rules that will be decisive individually for each conclusion in which they participate, but not constituying a "hard core" of the theory. The more courageous methods of reasoning have such a core, instead.

Given that only deductive rules determine strict preferences, it is clear that they will the exclusive members from the Collegium or the Oligarchy. This result shows that there are only two strata in a default theory: deductive rules(including axioms) and default rules.

It can be speculated that proper default rules can form also a many sorted set, given the different relevance of rules. Formalizations of this idea require to rank the default rules as a first step to perform reasoning, as it is acknowledged in [3].

V - Examples:

As an example of this method lets consider the case of Opus the penguin. The theory consists of the following rules:

I) penguin(x) ===> bird(x)

II) bird(x) ---> flies(x)

III) penguin(x) ===> -flies(x)

IV) penguin(Opus)

where ===> represents "strict" deductive implication and ---> represents "weak" default implication. The sets of maximaly consistent conclusions are:

A= {penguin(Opus), bird(Opus), flies(Opus)}

B= {penguin(Opus), bird(Opus), -flies(Opus)}

To find the global set of conclusions, consider the preference relations between A and B given by rules I to IV:

AI(I)B , AR(II)B , BP(III)A , AI(IV)B

In this case, Paretian optimality provides:

BRA

That means that B is prefered to A, due to axiom III, the only one that can determine a strict preference relation. Therefore, the new set of conclusions includes the evidence of Opus not flying and excludes the tentative but wrong conclusion that it does. Considering, as another example, the famous "Nixon diamond", where the rules are:

I) Republican(x)---->-Pacifist(x)

II) Quaker(x)------->Pacifist(x)

Given the evidence that Nixon is Republican and Quaker the corresponding axioms are:

III) Republican(Nixon)

IV) Quaker(Nixon)

Then the conclusions set plus the evidence will be:

{Republican(Nixon),Quaker(Nixon),Pacifist(Nixon),-Pacifist(Nixon)}

To avoid inconsistency and lacking additional information, the method of Paretian optimality will consider only subsets without contradictions. The sets to be considered will be A = {Republican(Nixon), Quaker(Nixon), Pacifist(Nixon)} and B = {Republican(Nixon), Quaker(Nixon), -Pacifist(Nixon)}. Then the following relations can be valid:

BR(I)A, AR(II)B, AI(III)B, AI(IV)B

Given the tie score between A and B the method can only conclude:

AIB

which is an expectable result knowing the equivalence of Paretian Optimality and theories with stable extensions this method cannot decide for one of the sets. Considering the method based in the Pareto principle only one set is prefered, because it is the only weakly prefered by all rules:

{Quaker(Nixon),Republican(Nixon)}

which is another expectable result considering the equivalence between the Pareto principle and skeptical inheritance.

VI - Conclusions:

Another rationale for coping with default reasoning has been presented in this paper, considering defaults as a kind of inductive rules and,emphasizing the tentativity of infered conclusions. Besides, a notion of model of default theories (inductive theories with elements of commonsense) with an epistemological bent was examined as preferable to a logical one.

To formalize the beforesaid concepts several tools borrowed from Social Choice theory have been useful. Two main results have been derived:

1) All forms of default reasoning can be seen as succesive constraints to an universal method of interpretation of default theories.

2) The set of classical rules (including evidences) is the decisive group on the conclusions of default theories.

Result 2 indicates that the methods of default reasoning are less "robust" than the scientific theories, as seen by the structuralist school of epistemology: given that this postkuhnian view admits in every theory several strata of stability, more than two kinds of formulations are decisive to the inferred conclusions. Perhaps an approach as considered in [16] may cope with the acknowledged problem of the ranking between defaults.

A way to strengh default theories is to use probabilistic semantics as proposed in [12]. Given that Decision Theory provided the tools to obtain both results of this paper, it is useful to look in it for other means to solve this problem. An attractive notion is that of expected utility, that allows to evaluate the relative importance of rules with respect to new evidence, and which also deserve further research.

ACKNOWLEDGMENT: The authors wish to thank Prof. Dr. Guillermo Simari for his helpful comments on an earlier draft of this paper.

VII - References:

[1] Bobrow, D.
Editor's Preface
Artificial Intelligence 13:1,2 (1980).

[2] Doyle, J.
On universal theories of defaults
Technical Report CMU-CS-88-111 (1988)

[3] Geffner, H.
Default Reasoning: Causal and Conditional Theories
Ph. D. Thesis.UCLA TR 137, (1989)

[4] Heal, G.
The theory of economic planning
Amsterdam, North-Holland (1973)

[5] Intriligator, M.
Mathematical optimization and economic theory,
N.Y., Prentice-Hall (1971).

[6] Israel, D.
What's wrong with non monotonic logic?
Proceedings of the First National Conference on AI (1980).

[7] Kuhn, T.
The structure of scientific revolutions
Chicago, The University of Chicago Press, (1962).

[8] Luce, R.- Raiffa H.
Games and decisions
N.Y., Wiley&Sons (1957).

[9] McCarthy, J.
Circumscription, a form of nonmonotonic reasoning
Artificial Intelligence 13:1, 2 (1980).

[10] McDermott, D.- Doyle, J.
Nonmonotonic logic I
Artificial Intelligence 13:1,2 (1980).

[11] Moore, R.
Semantical considerations on nonmonotonic logic
Artificial Intelligence 25, 1(1985)

[12] Pearl, J.
Probabilistic semantics for default reasoning
Technical Report. UCLA-CSD-870052 (1987).

[13] Perlis, D.
On the consistency of commonsense reasoning
Comput. Intell. 2 (1986).

[14] Popper, K.
The Logic of Scientific Discovery
London, Hutchinson & Co., (1959)

[15] Reiter, R.
A logic for default reasoning
Artificial Intelligence 13:1,2 (1980).

[16] Rescher, N.
Plausible reasoning
Amsterdam, Van Gorcum, (1976)

[17] Sen, A.
Social choice theory
in *Handbook of Mathematical Economics*
Arrow, K.-Intriligator, M. eds., N.Y., Elsevier (1986).

[18] Shoham, Y.
Reasoning about change: time and causation from the standpoint of Artificial Intelligence
Yale Univ. Ph.D. thesis (1986).

[19] Touretzky, D.- Horty, J.- Thomason, R.
A clash of intuitions: the current state of nonmonotonic multiple inheritance systems
Proceedings IJCAI-87 (1987).

UM SISTEMA PARA O TRABALHO COOPERATIVO BASEADO EM REDE LOCAL

Luiz Fernando Bier Melgarejo (1)
André Luiz Costa Ballista (2)
Márcio Quintaes Marchini (2)

Gruppoo- Grupo de Pesquisa em Programação Orientada a Objetos
Edugraf- CEC - CTC - Universidade Federal de Santa Catarina
e-mail: gruppoo@brufsc.bitnet

RESUMO

Os microcomputadores oferecem excelentes oportunidades para o trabalho cooperativo. A disseminação de redes locais abre instigantes perspectivas para o projeto de sistemas que estimulem o trabalho em grupo e criem ambientes ricos onde as pessoas possam se comunicar sem interferência e burocracia.

O objetivo desse artigo é apresentar um sistema de suporte ao trabalho cooperativo, baseado em rede local de microcomputadores. O sistema utiliza as técnicas de hipertexto\hipermídia para manipulação de uma base de dados distribuída, composta de documentos estruturados. A idéia de quadro-mural é utilizada como metáfora para um ambiente de trabalho individual e coletivo, como modelo para a interface de usuário e como método de particionamento lógico da base de dados.

ABSTRACT

Microcomputers are an excellent stimulus for cooperative problem solving. The dissemination of technologies in local area networks provides opportunities to the design of 'groupware',building rich environments where people can communicate without interference and bureaucracy.

This paper describes a system, based on a local network of computers, that supports cooperative work. The system uses hypertext/hypermedia concepts to manipulate a distributed database, consisting of structured documents. It uses a kind of 'bulletin board' metaphor as a framework to individual and shared work, as a model for the user interface and as a method for the logic fragmentation of the database.

(1) Professor Adjunto do Departamento de Ciências Estatísticas e da Computação
(2) Curso de Bacharelado em Ciência da Computação da UFSC

1. INTRODUÇÃO

A experiência tem mostrado que os computadores oferecem excelentes oportunidades para as atividades cooperativas, ao contrário do que temiam os oponentes iniciais da tecnologia. Nas aulas de laboratório de programação onde os estudantes usam microcomputadores, por exemplo, existe mais comunicação e solução conjunta de problemas do que nas aulas expositivas.

Talvez os estudantes trabalhem juntos porque o computador, sendo novidade, contribui para que eles esqueçam comportamentos enraizados que inibiam a colaboração, ou talvez isso aconteça pelo simples fato de que existem tão poucos micros que o compartilhamento se torna compulsório. Qualquer que seja a razão, temos aí uma oportunidade a ser ampliada.

Os estudantes estão predispostos naturalmente a pensar de maneira inovadora, mas necessitam ferramentas que lhes dêem suporte e encorajamento. Para isso são necessários ambientes ricos onde a informação possa fluir livremente e as pessoas possam colaborar e se comunicar sem interferência e burocracia.

A paulatina disseminação das redes locais de microcomputadores oferece instigantes oportunidades, e devemos arriscar propostas para as novas situações que se apresentam, projetando sistemas que tenham como meta a atividade cooperativa.

O objetivo desse artigo é apresentar um sistema de suporte computacional ao trabalho cooperativo, baseado em rede local de microcomputadores, que estamos desenvolvendo no Laboratório de Software Gráfico e Educacional (EDUGRAF), do Departamento de Ciências Estatísticas e da Computação (CEC) da UFSC. Este sistema, que estamos denominando hiperNet, utiliza extensivamente as técnicas de hipertexto\hipermídia [Conklin87], para manipular documentos estruturados organizados numa base de dados distribuída.

O desenvolvimento de sistemas de colaboração envolve basicamente três domínios distintos, que devem ser utilizados conjuntamente: banco de dados, comunicação e interface.

Banco de dados: o banco de dados deve servir principalmente como memória compartilhada e arquivo (de documentos, propostas,...). Deve incorporar facilidades para o controle de acesso, assim como de concorrência, necessárias num ambiente multiusuário.

Em nosso sistema, os (hiper)documentos são constituídos de (hiper)páginas de tipo texto, gráfico ou executável. As (hiper)páginas, ligadas entre si a critério do usuário, formam a base de dados hipermídia. Essa base de dados pode ser fácil e rapidamente explorada, para leitura, execução, navegação, edição. Consultas à base de dados podem ser feitas por nome ou por palavra-chave. Podem existir tantos (hiper)documentos quanto o espaço em discos permitir, distribuídos entre qualquer número de 'servidores', sendo sua localização física indiferente para o usuário.

As autorizações de acesso são baseadas em um modelo de fácil compreensão e o tratamento da concorrência é feito de forma a permitir que vários usuários possam acessar simultaneamente o mesmo (hiper)documento com o mínimo de restrições.

Comunicações: as comunicações podem ser classificadas em síncronas e assíncronas. Participantes nas comunicações síncronas estão conectados um ao outro por proximidade (numa sala) ou por tecnologia (ao telefone), enquanto que na comunicação assíncrona, a conexão entre os participantes é indireta (como em caixas-postais) e as comunicações são recebidas algum tempo depois de enviadas. Um ambiente de colaboração eficaz precisa incorporar esses dois tipos de comunicação.

Um ambiente que tem sido proposto como modelo de comunicação em sistemas cooperativos é o escritório (de projetos): os trabalhadores, a partir de seus postos de trabalho, ligam seus 'conectores de mídia', que devem ficar abertos durante o dia de trabalho. A qualquer momento, um trabalhador pode 'procurar' um colega e fazer-lhe uma pergunta, e\ou ambos podem fazer rascunhos ou anotações em telas que os dois possam observar [Vertelney90].

No domínio assíncrono, os sistemas de "e-mail" vão se tornando populares, e pode-se utilizar como modelo as tecnologias já desenvolvidas para organizar os serviços de troca de mensagens, como as da Bitnet.

Neste sistema, propomos duas formas principais de comunicação interpessoal: o hiperBilhete, para comunicações simples e esporádicas, e a hiperMensagem, para troca de documentos estruturados. Existe ainda outra forma possível, para comunicações persistentes por compartilhamento da base de dados, através de um hiperMural.

Interfaces de Usuário: as interfaces de usuário existentes foram quase sempre projetadas visando usários individuais, trabalhando sozinhos. E preciso pesquisar novos modelos que suportem com naturalidade diversos usuários, assim como atividades orientadas a grupos.
O que parece estar se tornando claro é que as técnicas de hipertexto tendem a se tornar um padrão para o uso do computador enquanto mídia -meio de comunicação- (1).

Uma das metáforas de interface mais conhecidas é a "desktop" [Apple87]. Pensamos que essa idéia está comprometida com ambientes voltados ao trabalho individual, e exploramos um modelo alternativo, usando como metáfora o quadro-mural, este sim tradicionalmente associado no cotidiano com a comunicação entre pessoas e grupos.

Na hiperNet o quadro-mural foi enriquecido com as técnicas de hipertexto (hiperMural) e, além de servir como contexto individual de trabalho, possibilita um modo compreensível de controlar o acesso às áreas compartilhadas, constituindo ainda um ambiente para a comunicação pública da rede, no hiperMural Coletivo.

A seguir, apresentamos a abordagem utilizada para a criação e desenvolvimento do sistema (item 2); os elementos básicos de hipertexto e sua organização através do modelo de hiperMurais (item 3); os objetos que modelam funcionalmente o sistema (item 4); o sistema de correspondência (item 5); características da interface do usuário (item 6); um cenário de aplicação e conclusões atuais (item 7); a bibliografia e as

(1) "Hypertext is a tool, brought to the personal computer, that offers the beginnings of a technical grammar for the medium: not shots and cuts, but buttons and links...The medium itself begins to coalesce, find form, find its true nature. And the true nature of the medium is not to recreate the novel, nor to disguise narrative as a game, nor to pretend ...

referências pertinentes (ítem 8).

2. ABORDAGEM UTILIZADA

Numa abordagem tradicional de redes locais, a conexão entre os equipamentos tem como objetivo básico o compartilhamento de equipamentos (discos rígidos, impressoras, etc.), considerados os recursos escassos do sistema. Nessa abordagem, a intercomunicação possível entre os usuários é uma decorrência (já que o hardware está conectado). A abordagem que estamos pesquisando é orientada-ao-usuário, no sentido de vê-lo como o elemento mais importante do sistema, e objetiva favorecer o compartilhamento dos mais nobres recursos circulantes na rede: as idéias.

Essa visão tende a se generalizar, na medida do avanço da área de redes, conforme se depreende da literatura [Tarouco87] [Tanenbaum88]. Um de nossos objetivos de projeto é verificar até que ponto esse enfoque é praticável hoje, utilizando equipamentos pessoais (PC-compatível) conectados através de redes locais de baixo custo. Buscamos, consideradas essas restrições, uma solução de compromisso que combine a riqueza de possibilidades das técnicas de hipertexto com a eficácia de um sistema de comunicação em rede.

Para isso, implementamos um sistema de rede local de custo ínfimo (conexão através de par trançado), com topologia em barramento. Acima do nível de enlace dessa rede, implementamos um sistema básico de desenvolvimento, compatível com sistemas de rede disponíveis no mercado nacional. Essas duas etapas estão detalhadas em outros trabalhos [Marchini90] [Marchini91]. Sobre essa base, estamos desenvolvendo o software aqui descrito, usando nossa rede e alternadamente uma rede comercial. Toda a programação é na linguagem Modula2, utilizando uma biblioteca de módulos genéricos e técnicas de programação modular baseada em objetos desenvolvidas pelo Gruppoo.

(1) cont.: ...is cinema or television, but to be itself, its own graphic,textual, sonic, interactive self. "
Rob Swigart, in [Laurel90], pag.145

3. O MODELO hiperMural

Um quadro-mural é basicamente um meio de comunicação, normalmente usado de modo assíncrono. É utilizado rotineiramente por grupos de afinidade, envolvidos por um interesse comum. Em certos ambientes, e para certos usos, pode ser considerado um mecanismo coletivo, público, tal a generalidade dos assuntos afixados e o número de pessoas que a ele têm acesso. De outro lado, pode ser às vezes encontrado pendurado em frente a um posto de trabalho individual, sendo utilizado aí como um quadro de lembretes no estilo agenda pessoal.

Num quadro-mural podemos encontrar vários tipos de itens (objetos de informação): pequenos textos, fotocópias, cartazes, recortes de jornais, gráficos, etc...

Em nosso projeto, o quadro-mural está associado com as técnicas de hipertexto, num modelo que chamamos hiperMural. Um hiperMural (hM) é constituído de um conjunto de hiperDocumentos e um grupo de autores. O grupo de autores é um conjunto de identificadores dos usuários que estão 'cadastrados' naquêle hM. Somente os usuários membros do grupo de autores do hM têm acesso ao seu conjunto de hiperDocumentos.

Os itens de informação possíveis de serem 'afixados' num hiperMural foram modelados através de duas classes de objetos que chamamos hiperPáginas (hP) e anotações. As hP's podem ser de três tipos: hP-texto, hP-gráfica (contém imagens digitalizadas) e hP-executável (contém instruções que podem ser executadas pela máquina a critério do usuário).

Uma hiperPágina pode estar associada a uma ou várias outras hiperPáginas, através de ligações hierárquicas, formando hiperDocumentos. Um hiperDocumento é identificado por um nome único no seu hiperMural e tem uma hiperPágina inicial, que é a primeira hP criada em seu nome.

Um trecho qualquer de uma hiperPágina pode ser associado a um texto, acompanhado de identificação do autor e data/hora, que chamamos anotação. É possível ligar também uma anotação, relacionando-a com uma outra hiperPágina, possivelmente noutro hiperDocumento, caracterizando uma referência.

A figura 1 representa os principais elementos encontrados num hiperMural.

O conjunto das hiperPáginas e anotações pertencentes a um hiperMural forma seu hiperEspaço. Ao ato de percorrer as informações através das ligações existentes no hiperEspaço chamamos navegação.

Ao utilizar o quadro-mural como um modelo conceitual para o desenvolvimento e apresentação do sistema, esperamos propiciar um uso efetivo por parte dos usuários, que poderão utilizar a idéia como modelo mental. Um modelo mental é uma representação cognitiva que auxilia o usuário a familiarizar-se mais facilmente com um sistema e fazer previsões sobre o seu comportamento. Os modelos mentais permitem tanto entender melhor as situações-problema como prever as conseqüências das ações examinadas para sua solução [Marchionini88].

O modelo hiperMural é utilizado também para que o usuário possa dispor de vários contextos de trabalho, organizados de forma compreensível e flexível. Existem três classes de contextos possíveis: hiperMural Individual, hiperMural de Grupo e hiperMural Coletivo.

- HiperMural Individual

Todo usuário, ao adquirir acesso à hiperNet, tem estabelecida para si uma área de trabalho, chamada hiperMural Individual (hMI). O objetivo do hMI é fornecer uma área de trabalho onde a privacidade do indivíduo esteja garantida. Todos os hiperDocumentos criados num hMI ou trazidos para ele por cópia são totalmente protegidos de outros usuários, mesmo para operações não destrutivas como navegação ou visualização de nomes de hiperDocumentos.

Os hMI estimulam o usuário a desenvolver sua produção individual de documentos utilizando o sistema, o que acaba favorecendo a comunicação interpessoal ou coletiva, na medida em que seu trabalho é, desde o início, completamente compatível com o sistema compartilhado. Assim, a qualquer momento o usuário pode enviar seu hiperDocumento a um colega, solicitando por exemplo uma revisão (via anotações), ou levar seu hiperDocumento para o hiperMural de um grupo que tenha interesse no assunto para desenvolvê-lo em conjunto. Isso não é o

que acontece nos sistemas tradicionais de correio eletrônico, por exemplo (onde existe a necessidade de utilitários de conversão, exportação e importação de um trabalho individual para o sistema de comunicação), o que acarreta freqüentemente menos colaboração.

- HiperMural de Grupo

Para o trabalho em grupo, existem os hiperMurais de Grupo (hMG), que possibilitam o acesso simultâneo de vários usuários. Qualquer usuário, a qualquer momento, pode criar um hiperMural de Grupo, bastando dar-lhe um nome e informar quais os outros usuários que irão participar desse hMG. Esses usuários serão informados pelo sistema, e tanto o criador como os outros terão a partir daí iguais direitos sobre o hiperMural em termos de criação, destruição ou edição de hiperDocumentos.

- HiperMural Coletivo

O hiperMural Coletivo é aquele onde os autores são todos os usuários hiperNet, configurando uma espécie de BBS ("Bulletin Board System") em hipermídia. Os hiperDocumentos do hiperMural Coletivo podem ser acessados livremente por qualquer usuário da rede, com iguais direitos de criação, destruição, edição, ligação, etc.

O hiperMural Coletivo pode ser o ambiente apropriado para duas classes de aplicações que desejamos incentivar: a) evolução de hiperDocumentos de criação coletiva, para livre discussão de tópicos de interesse da comunidade ; b) jornalMural: notas, compra e venda, convites p/eventos, etc... Esperamos fornecer um utilitário que aceite comandos de indexação por palavras chave e por data. Esse utilitário seria rodado automaticamente pelo sistema de forma periódica (semanal, por exemplo). O produto é o jornalMural, um hiperDocumento indexando os assuntos da semana, a partir de um conjunto de tópicos escolhidos pelos usuários.

- Controle de Concorrência

E possivel a navegação simultânea nos hiperMurais compartilhados, de modo que se pode ter mais de um usuário explorando um mesmo hiperDocumento. O controle de concorrência para fins de edição se dá a nivel de hiperPágina. O primeiro usuário que acessa uma hP tem garantidos os direitos de edição até sair dela. Os demais leitores são informados automaticamente dessa 'reserva de direitos' temporária quando entrarem na mesma hiperPágina.

4. ESQUEMA FUNCIONAL

O sistema hiperNet é composto de seis elementos funcionais, que se combinam em algumas configurações, de acordo com a disponibilidade de dispositivos de armazenamento e comunicação. Apresentamos a seguir uma descrição desses elementos e configurações (figura 2).

- Guardião de HiperMurais

Um guardião de hiperMurais (ghM)tem sob sua responsabilidade um conjunto de hiperMurais. Sua função é organizar o acesso a esses hiperMurais, fornecendo informações sobre seu conteúdo. Recebe consultas internas do agente de Usuário local e consultas externas, feitas por outros agentes de Usuário através do agente de Comunicações. Atende aos pedidos um por vez, colocando-os numa fila de atendimento. Para responder às consultas que recebe, ativa o servidor de Arquivos. As estações da rede que não fornecerem área em disco para armazenamento não instanciarão nenhum guardião.

- Servidor de Arquivos

A função do servidor de arquivos (sA) é monitorar o acesso à memória secundária que for alocada como área disponivel da estação. Recebe consultas dos guardiões de hiperMurais e do hiperServidor. O hS não é instanciado numa estação que não aloca área em disco para o sistema.

- HiperServidor

O hiperServidor (hS) tem como principal atribuição manter informações sobre o estado geral do sistema. Cabe ao hS manter o cadastro geral de usuários (senhas, hiperMurais acessíveis, etc.) e estações da rede assim como tabelas relacionando os guardiões com os hiperMurais que lhes foram alocados. E o hiperServidor que atribui a um guardião a responsabilidade sobre um hiperMural qualquer, de acordo com a política de alocação praticada na instalação.

No início de uma sessão, depois de admitido o acesso de um usuário, o hS irá fornecer à estação que abriu a sessão informações para o agente daquele específico usuário (hiperMurais aos quais ele tem acesso, quais os guardiões desses hM's, informações sobre a última sessão do usuário no sistema).

Durante uma sessão, para acesso à base de dados, cada agente de usuário passa a se comunicar com cada um dos guardiões, na medida em que o usuário for trocando de contexto, ou seja, passando de um hiperMural a outro. Isto evita consultas freqüentes ao hiperServidor e distribui as tarefas na rede.

O hiperServidor (hS na figura 2) é instanciado em apenas uma estação da rede, embora possa vir a ser replicado para que o sistema tolere uma possível queda da estação hospedeira.

- Agente de Comunicações

Cabe ao agente de Comunicações (aC) permitir a troca de informações entre entidades remotas e fazer o roteamento de mensagens entre as entidades locais.

O cliente (aquele que lhe pede informações) de um aC é um agente de usuário. Para fornecer a resposta adequada, ele se comunica com o hiperServidor (hS) e/ou com o guardião de hiperMurais (ghM). Caso a informação esteja na própria estação, o aC simplesmente se comunica com a entidade local relevante (hS e/ou ghM) e devolve a resposta para o agente de usuário. Se, ao contrário, a informação procurada estiver loca-

lizada numa estação remota, é utilizado o sistema de rede para que estas estações possam trocar informações.

Duas entidades aC trocam informações através do protocolo especifico da aplicação. Esse protocolo é independente dos protocolos das camadas inferiores e somente os utiliza como suporte.

Um agente de Comunicações é instanciado mesmo para uma estação isolada, de forma a preservar a modularidade do sistema e a homogeneidade entre a versão-rede e a versão mono-estação.

- Agente de Usuário

Um agente de usuário (aU) é responsável pelo provimento de todas as informações necessárias à comunicação com o usuário (via interface). De um lado se comunica com o hiperServidor e com os guardiões de hiperMurais para o armazenamento e recuperação de informações. De outro lado se comunica com o agente de interface, fornececendo suporte às demandas surgidas a partir da interação deste com o usuário.

O agente de usuário mantém localmente o estado dos atributos de sessão, como: hora de Início, tempo de utilização, trilha de navegação, hiperMural atual, hiperDocumento atual, consultas e respostas, etc... Ao fim de uma sessão, vários desses atributos serão comunicados ao hS, para utilização em próxima interação do usuário com o sistema.

- Agente de Interface

O agente de interface (aI) controla toda comunicação direta entre o usuário e o sistema, atendendo os dispositivos de entrada (teclados, apontadores) e fornecendo 'feed-back' via monitor de vídeo. Ele é, assim, o responsável pela visualização que o usuário tem do sistema.

O agente de interface é composto de vários segmentos que vão tratar os estados modais da interação (início e fim de sessão, edição\ligação\navegação, manipulação de correspondência, tratamento das anotações, etc.). Compreende tambem a

manipulação de eventos, que recebe as entradas do usuário via "mouse", teclado ou outro dispositivo, fornecendo "feed-back" imediato ou passando informações aos outros módulos para tratamento.

- Configurações Aplicáveis

Várias estações conectadas em rede

a) Uma das estações vai conter o hiperServidor, que é o núcleo de informações do sistema. Provavelmente a mesma estação dispõe de área em disco alocada para o sistema, possuindo também ao menos um guardião de hiperMurais. Na figura 2, esta situação é representada pela estação 'H'.

b) Estações que oferecem área para armazenamento instanciam ao menos um guardião de hiperMurais. Não é instanciado o hiperServidor. Na figura 2, representada pela estação 'D'.

c) Estações que não alocam área em disco para uso do sistema. Não são instanciados nem o hiperServidor nem o guardião de hiperMurais. As necessidades de informação são transmitidas pelo agente de Usuário ao agente de Comunicações, que irá passá-las via rede para outro agente de Comunicações, na estação que contém a informação desejada. Representada na figura 2 pela estação 'S'.

Uma estação de trabalho isolada

Neste caso, não há rede. Todas as entidades são instanciadas localmente. O agente de comunicações faz o roteamento de mensagens para o hiperServidor e os guardiões de hiperMural.

A implantação em uma estação isolada pode ser utilizada: a) Como um software de uso pessoal, com facilidade de exportar hiperDocumentos para a versão-rede; b) Como ambiente multiusuário onde várias pessoas trabalhem em horários diferentes; c) Para desenvolvimento do sistema (é possível utilizar a saída serial 'em curto', de forma a testar até mesmo o sistema de rede num só equipamento).

5. SERVIÇO DE CORRESPONDÊNCIA

A hiperNET oferece serviços de correspondência interpessoal (ou de pessoa a grupo) na forma de hiperBilhetes e hiperMensagens.

- O HiperBilhete

Os hiperBilhetes (hB) são trechos de texto produzidos por um usuário, acompanhados pela data\hora e nome do remetente, adicionados pelo sistema. Os hB's servem para o envio de mensagens simples e curtas, de modo esporádico. Se o destinatário estiver ocupando alguma estação da rede no momento, o hiperBilhete será 'publicado' na linha superior de sua tela; caso contrário, será armazenado em sua 'caixa-postal'.

Os hiperBilhetes são também um modo do sistema se comunicar com o usuário, para informá-lo de eventos de interesse ou solicitar que ele forneça informações via teclado.

Um hiperBilhete pode estar ligado a uma hiperPágina, da mesma forma que os ítens de anotação. Isto permite que o sistema possa 'conduzir' o usuário, se este 'clicar' sobre o hiperBilhete, para alguma informação adicional ou auxiliar contida em uma hiperPágina produzida automaticamente.

- HiperMensagens

Uma hiperMensagem (hM) é um hiperDocumento, acompanhado de data\hora e nome do remetente, enviado de um usuário a outro da rede. Supre a necessidade de envio de mensagens mais complexas e estruturadas, quando não se deseja o compartilhamento através de hiperMurais.

O ambiente de edição\navegação oferece serviços auxiliares apropriados para hM's, semelhantes aos sistemas tradicionais de correio eletrônico.

6. CARACTERISTICAS DA INTERFACE DO USUARIO

O sistema apresenta uma janela de visualização de tamanho fixo, que emoldura a tela inteira. Essa janela é como um 'foco' sobre uma hiperPágina de um hiperDocumento. A medida que se navega, modifica-se a hP visualizada.

Na borda superior da moldura encontra-se o nome do hiperMural sendo navegado, o nome do hiperDocumento e o nome da hiperPágina atual. A borda inferior da moldura é constituída por uma barra de menus do tipo "pop-up", que permite a ativação de funções de apoio.

A interface é orientada a dispositivo apontador do tipo "mouse", passível de simulação via teclado. O cursor de navegação acompanha os movimentos do dispositivo apontador, fornecendo um escopo para várias operações de manipulação.

Os usuários navegam na base de dados indo de uma hiperPágina a outra através das ligações existentes. Para visualizar a hP que está conectada a um ítem, o usuário, com o cursor posicionado sôbre o ítem, 'clica' um botão do dispositivo apontador. Isso fará com que a hiperPágina atual seja substituída na janela de visualização pela hP que está conectada àquele ítem (ou que seja executado um programa qualquer, no caso de hP's executáveis).

E utilizado sempre que possível o princípio de manipulação direta [Apple87]: o utilizador primeiro seleciona o objeto sobre o qual deseja operar e depois o comando desejado. Como a posição do cursor já indica qual o ítem selecionado, resta a escolha da operação a ser realizada, normalmente com o uso de quatro opções do dispositivo apontador. Essa combinação permite que muitas vezes baste simplesmente um 'clic' no dispositivo apontador para selecionar uma operação juntamente com o ítem ao qual ela se aplica.

Nas hP's gráficas, os ítens de ligação são regiões retangulares ou circulares da imagem que o usuário marca e liga com outras hP's. Para não prejudicar a visualização, a identificação dos ítens 'sensíveis' é feita a pedido do navegador, e o mesmo é feito com as anotações.

E comum nos sistemas de hipertexto a utilização de 'botões' ("buttons") como origem das ligações, conectando uma região da tela com outra unidade de informação. Se o usuário 'apertar o botão', o sistema apresentará o conteúdo de informação que está conectado a êle, possivelmente numa nova janela.

No sistema hiperNet, no caso das ligações hierárquicas, o próprio texto faz o papel de botão. Assim, clicando-se o dispositivo apontador quando o cursor de navegação estiver sobre um ítem de texto, teremos a apresentação da hiperPágina a que ele está conectado.

8. CONSIDERAÇÕES FINAIS

- Um Cenário de Aplicação

Um sistema de natureza genérica como o apresentado pode ter diversas aplicações. Sumarizamos aqui um cenário proposto para o Centro Tecnológico da UFSC [Melgarejo89] envolvendo professor, sala de aula, laboratório e estudantes:

Professor: utiliza a hiperNet a partir de sua sala de trabalho, para produzir e organizar o material didático, que contém seu próprio texto, cópias, referências bibliográficas, gráficos (produzidos em qualquer editor gráfico) ou imagens digitalizadas encomendadas ao laboratório EDUGRAF (retiradas de outras fontes por apropriação direta via "scanners"). A organização será feita pela estruturação do material através da criação de ligações entre os fragmentos de conteúdo. O material poderá ser impresso segundo critérios de linearização e referenciação escolhidos (o que possibilita a produção de apostilas de forma semi-automática).

A sala de aula está equipada com microcomputador (conectado à hiperNet), retroprojetor e tela plana de cristal líquido. A apresentação do material didático é feita por retroprojeção da tela do computador. Durante a exposição, são utilizadas as ligações do hipertexto como um escalonador de conteúdos, ou seja, um roteiro dinâmico da aula. Dinâmico porque essas ligações, assim como o próprio conteúdo, podem ser alteradas durante a apresentação, a partir de sugestões de ouvintes ou de novos 'insights' do professor.

Os estudantes, utilizando a rede a partir de um laboratório de ensino, podem dispor do mesmo material instrucional estruturado que foi visto em sala de aula, para explorá-lo com fins de reforço ou anotação. Os alunos podem também fazer um projeto em grupo, por exemplo,produzindo seu próprio material hipermídia, utilizando as facilidades de comunicação, discussão coletiva, acesso à base de dados sobre o assunto, etc.. O hiperdocumento resultante pode ser enviado ao professor para revisão, através de uma hiperMensagem, ou ser discutido em conjunto com o professor, num hiperMural.

De todo modo, na sala de aula, para fins de apresentação final, o projeto do grupo será acessado via rede e apresentado utilizando a tela de cristal líquido e o retroprojetor. Se fôr o caso, outras pessoas poderão ter sido avisadas da apresentação através de convite (acompanhado possivelmente do mapa de localização da sala) no jornalMural.

- Resultados

Esse projeto tem nos trazido experiência nas diversas áreas envolvidas, ampliada pelo fato do sistema estar sendo desenvolvido no grupo de pesquisa, desde a conexão física dos equipamentos até a interface do usuário. O sistema aqui apresentado está descrito em mais detalhes em documentos de projeto.

A implementação tem se dado em várias frentes. Já completamos uma implementação-piloto [Ballista90], integrando serviços de rede NetWare com o sistema ConTexto, desenvolvido anteriormente pelo grupo [Hollanda89].

Atualmente (março de 91), dispomos de um conjunto de módulos de desenvolvimento que permite que um mesmo programa, já compilado, possa ser executado sem modificações, tanto na rede de custo ínfimo que implantamos em laboratório como numa rede local CETUS [Marchini91]. A interface está bastante adiantada, e pretendemos ainda neste semestre concluir um protótipo que implemente completamente o modelo apresentado neste artigo.

- Agradecimentos

Os autores registram agradecimentos a Sérgio Martini de Hollanda e Alberto J. Oliveira pela implementação e suporte do ConTexto, assim como ao pessoal do Laboratório de Controle e Microinformática (LCMI-EEL), que tem fornecido documentação e acesso à rede lá instalada. Agradecemos também aos demais colegas do Gruppoo e Edugraf, com os quais temos compartilhado um inquieto e descontraído ambiente de pesquisa.

8. BIBLIOGRAFIA E REFERÊNCIAS

Livros

[Apple87] 'Human Interface Guidelines: The Apple Desktop Interface'; Apple Computer, Inc.- Addison-Wesley Publishing Company, Inc. - 1987

[Laurel90] 'The Art of Human-Computer Interface Design'; Brenda Laurel, ed. - Addison-Wesley Publishing Company, Inc. - 1990

[Tanenbaum88] 'Computer Networks'; Andrew S. Tanenbaum; Prentice-Hall International, Inc. - 1988

[Tarouco86] 'Redes de Computadores Locais e de Longa Longa Distância'; Liane Tarouco, Ed. McGraw-Hill, Ltda. - 1986

[Tecnologia88] 'Tecnologia e Liberdade'; Comunicações do Colóquio Internacional de Estudos sobre Tecnologia e Liberdade - Lisboa - 87; ed. Sementeira, CRL - 1988

Artigos e documentos

[Ballista90] 'Integração do sistema ConTexto com a rede NetWare'; André Luiz Costa Ballista - Relatório Técnico 90/012 - Gruppoo-Edugraf-CEC-CTC-UFSC - 1990

[Conklin87] 'Hypertext: An Introduction and Survey'; Jeff Conklin - Computer IEEE, pags. 17-41 - september 1987

[Hollanda89] 'Projeto ConTexto'; Sérgio Martini de Hollanda - Relatório de Projeto Final - Curso de Bacharelado em Computação - UFSC - 1989

[Marchini90] 'Metodologia orientada a objetos na implementação de uma Rede Local de custo infimo' ; Márcio Marchini e Gláucio Schumacher; Relatório Técnico 90/013 - Gruppoo-Edugraf-CEC-CTC-UFSC - 1990

[Marchini91] 'GruppooNet: Um sistema básico de desenvolvimento de software multiusuário'; Márcio Marchini - Relatório Técnico 91/001 - Gruppoo-Edugraf-CEC-CTC-UFSC - 1991

[Marchionini88] 'Finding Facts vs. Browsing Knowledge in Hypertext Systems'; Gary Marchionini and Ben Shneiderman - Computer IEEE - January 1988

[Melgarejo89] 'Projeto CTC90'; Luiz Fernando Bier Melgarejo e Luiz Fernando Jacinto Maia - Centro Tecnológico-CEC-UFSC - 1989

[Vertelney90] 'An environment for collaboration'; Harry Vertelney; in [Laurel90] , pp. 161-169

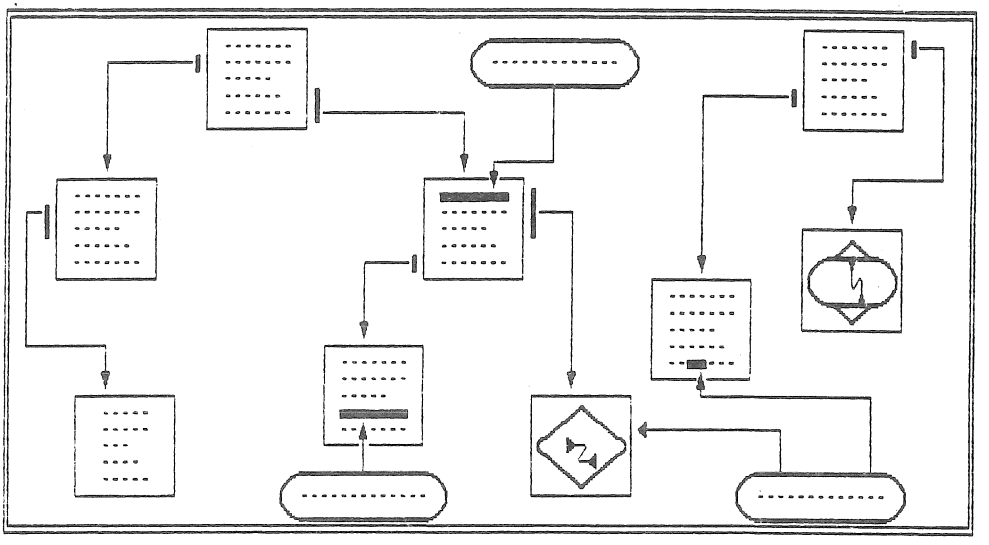


Figura 1 : Elementos de um hiperMural

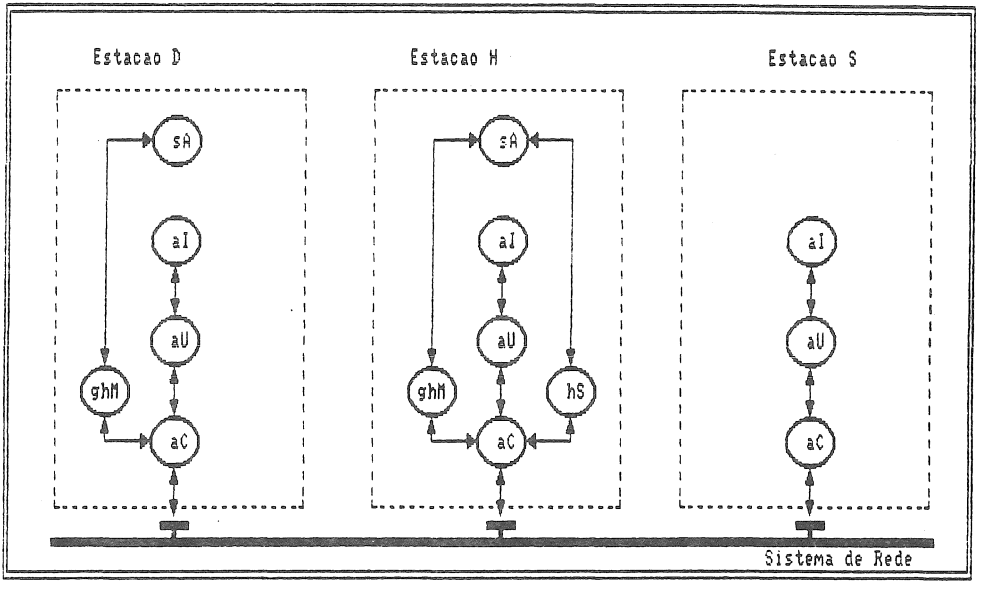


Figura 2 : Esquema Funcional

IV CONFERENCIA VENEZOLANA
DE INTELIGENCIA ARTIFICIAL Y SISTEMAS EXPERTOS

CNIASE '91

EXCAYA: AYUDA EXPERTA EN CARACTERIZACION DE YACIMIENTOS

LAGOVEN, S.A.
DEPARTAMENTO DE PRODUCCION

Caracas, Venezuela
Julio 1991

MOTIVACION

El simulador numérico de yacimientos es una herramienta que permite al ingeniero predecir el comportamiento de los yacimientos sometidos a diferentes esquemas de producción. Así como un yacimiento sólo puede producirse una vez, un modelo permite obtener resultados aproximados en diferentes esquemas de desarrollo, lo cual ayuda a seleccionar las condiciones óptimas de explotación.

El simulador tiene la capacidad de representar detalles geológicos, petrofísicos y operacionales del yacimiento, y puede predecir el recobro de hidrocarburos si los datos empleados en el modelaje describen adecuadamente su configuración geométrica, las propiedades de la roca y de los fluídos y los mecanismos que controlan sus desplazamientos.

Los datos inicialmente asignados en el simulador representan el mejor estimado sobre los parámetros que describen el yacimiento. Este conjunto de datos será modificado en varias oportunidades hasta lograr reproducir, en un grado "aceptable", el comportamiento histórico ("History Match"), durante el cual se logra la caracterización del yacimiento.

En este marco de modelaje de yacimientos se ha desarrollado la aplicación computarizada EXCAYA para que el usuario, a través de un diálogo, reciba apoyo en el proceso iterativo de ajuste histórico, y también sirva de guia en la selección del simulador adecuado, cuando sea pertinente su empleo.

ITERACION EN MODELAJE MATEMATICO

No es posible establecer un procedimiento estándar para cotejar con agilidad el comportamiento histórico. Cada situación es única y amerita experiencia, de acuerdo al área geológica de estudio y simulador empleado, para deducir los mecanismos que contribuyen al comportamiento del yacimiento y para seleccionar las variables idóneas en la manipulación del simulador.

Este proceso consume la mayor parte del tiempo asignado a un estudio modelado a través del simulador, y se carateriza por un lazo iterativo en el que el ingeniero reformula su concepción del yacimiento, como resultado de las respuestas del simulador a los parámetros que está usando para describirlo.

Al revisar el efecto ocasionado por los cambios en el intento N, el ingeniero escoge los datos de entrada a emplear en el intento N + 1, y debido a la cantidad las variables que interactúan, pudiera no captar adecuadamente la relación causa/efecto, y posteriormente realizar corridos innecesarios, con el consecuente incremento en costo del estudio.

En la Figura No.1 se representa en forma esquemática la naturaleza cíclica del cotejo. En particular, al imponer en el simulador la tasa histórica de producción medida, se observa el comportamiento de las relaciones agua/petróleo y gas petróleo, así como el comportamiento de la presión.

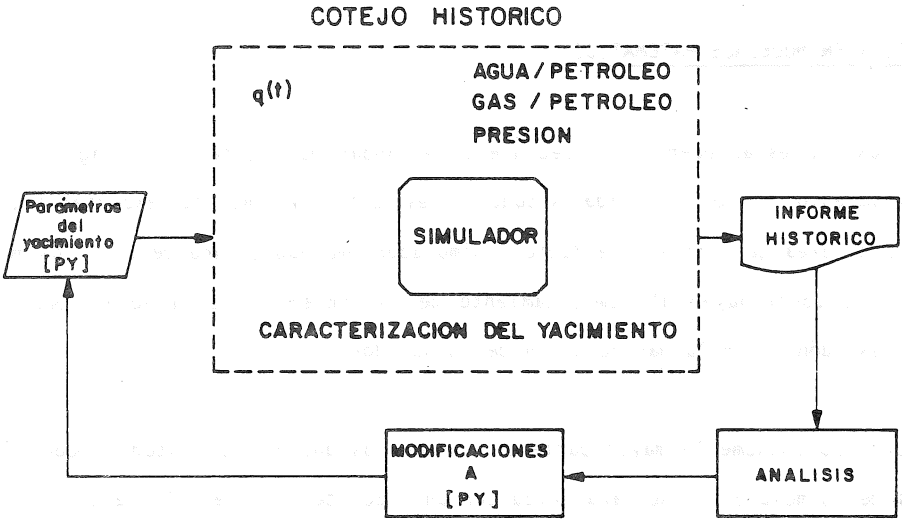


Figura N° 1

Al tratar de reproducir la presión y las tasas de agua y de gas, a través de sucesivos ajustes de parámetros identificadores del yacimiento, se va logrando la caracterización de la acumulación de hidrocarburos, cuya futura explotación se desea planificar de la mejor manera posible.

DESCRIPCION DE EXCAYA

Con el propósito de brindar apoyo en la tarea de simulación al ingeniero de yacimientos, se ha diseñado EXCAYA con dos temas principales: SIMUL y YACIM, tal como se aprecia en el diagrama del Anexo.

- En el ambiente de software, SIMUL se compone de dos módulos:

  ° En el primero, SELEC, a través de un diálogo con el usuario, EXCAYA llega a recomendar el simulador cónsono con el marco de referencia a un modelaje planteado. SELEC ha sido implantado con tecnología de sistemas expertos.

  ° En el segundo, INVEN, el usuario verifica los simuladores disponibles en la empresa, sus alcances y los controles para su ejecución. Aquí se manejan descripciones en texto.

- En el ambiente de caracterización del yacimiento, YACIM, se desglosa en tres módulos.

  ° En MACRO, se reciben indicaciones de cómo proceder iterativamente en las etapas clásicas de ajuste histórico. Aquí están plasmadas metodologías aceptadas en el ambiente de simulación y se presentan a través de textos y diagramas.

  ° En DIAG se ofrece un menú de casos con recomendaciones heurísticas ante situaciones resultantes de seguir las indicaciones de MACRO. A través de diagramas básicamente, están representadas experiencias vividas en modelaje de yacimientos.

  ° DICIO simplemente proporciona descripción de las variables que intervienen

en los estudios de simulación, junto con notas alusivas a las repercusiones de cambiar sus valores, en la descripción del yacimiento.

MODULOS DE EXCAYA BASADOS EN CONOCIMIENTO

De acuerdo a la descripción anterior, éstos se identifican como SELEC, MACRO y DIAG.

SELEC:

En SELEC, el ingeniero navega a través de un diálogo, en el cual se contempla:

(i) La ubicación del usuario en el escenario de simulación; grado de conocimiento en esta actividad, así como recomendaciones para refrentar el estudio del yacimiento.

(ii) Factibilidad de emplear un simulador en el estudio planteado. Si no luce oportuno emplear un simulador, se entra en otro diálogo para escoger el método convencional de cálculo que se debe utilizar.

(iii) Si, como resultado afirmativo de (ii), o por razones ajenas, es necesario realizar el estudio con un simulador, se procesa otra interacción a través de una cadena de preguntas y respuestas. Al responder afirmativamente cualesquiera de ellas, se obtiene la recomendación apropiada para utilizar uno o varios simuladores.

Al responder negativamente toda la cadena, se llega a un punto donde se contemplan las opciones micro y mainframe como alternativas, y ante la respuesta escogida, se dan recomendaciones oportunas sobre los simuladores disponibles.

En la rama de EXCAYA donde se manejan los conceptos asociados a yacimientos, se entiende que el usuario está utilizando uno de los nueve simuladores de la empresa, y en ella se ubican los módulos MACRO y DIAG (véase Anexo).

MACRO:

En MACRO, para dar una idea de su contenido, se presenta en la Figura No.2 el caso del esquema iterativo pertinente al ajuste de presión durante el cotejo histórico. En él se aprecia sus conexiones con el paso anterior (I), de Inicialización, y el paso siguiente (III) de Saturación.

En el ambiente de este módulo, el usuario puede reubicarse en la opción de texto asociada, donde se desglosa el contenido del flujograma.

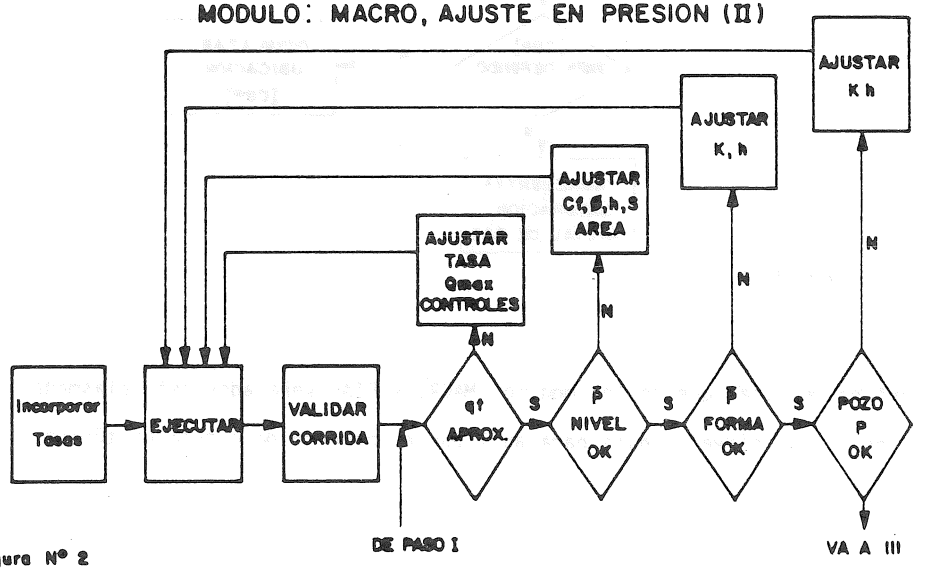


Figura N° 2

DIAG:

En el módulo de diagnóstico heurístico, DIAG, se ofrecen recomendaciones ante situaciones que se presentan al seguir las indicaciones pautadas en el módulo MACRO. Así en la Figura No.3, a título de ejemplo, se presenta un esquema de la respuesta a una consulta sobre el inconveniente de obtener en una corrida de cotejo un volumen de gas en sitio menor que el determinado previamente para el yacimiento.

MODULO HEURISTICO (DIAG), SECCION SATURACIONES

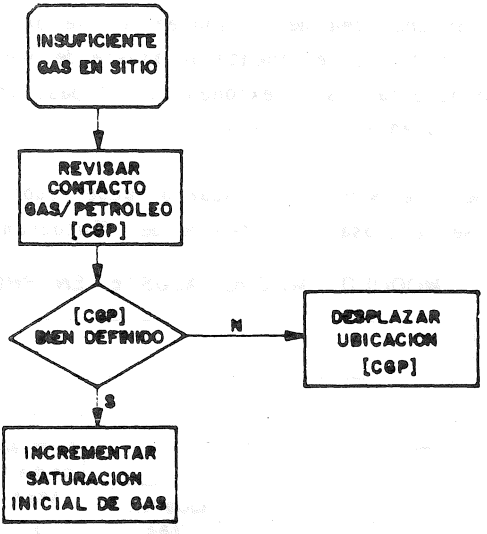


Figura N° 3

Es oportuno resaltar que en los módulos MACRO y DIAG señalados está plasmada experiencia, que sirve de guia para agilizar el proceso de cotejo histórico.

EXCAYA ha sido implantada como una ayuda didáctica para ingenieros que se inician en el ambiente de simulación de yacimientos, o para otros que practicaron esta disciplina en el pasado y vuelven a ella.

EXCAYA ha sido desarrollada bajo la modalidad de tesis de grado de Nicelia Hidalgo y Yineira Montilla para optar al título de Licenciado en Computación en la Facultad de Ciencias de la Universidad Central de Venezuela, 1991.

BIBLIOGRAFIA

"Artificial Intelligence in Business. Expert Systems", Paul Harmon, David King, 1985. John Wiley.

"A Guide to Expert Systems", Donald A. Waterman, 1986. Addison - Wesley.

"Knowledge Acquisition. Principles and Guidelines", Karen L. McGraw, Karan Harbison - Briggs, 1989. Prentice-Hall.

"Modern Reservoir Engineering - A Simulation Approach", Henry B. Crichlow, 1977 Prentice - Hall.

"Level 5 User's Guide", 1990. Information Builders.

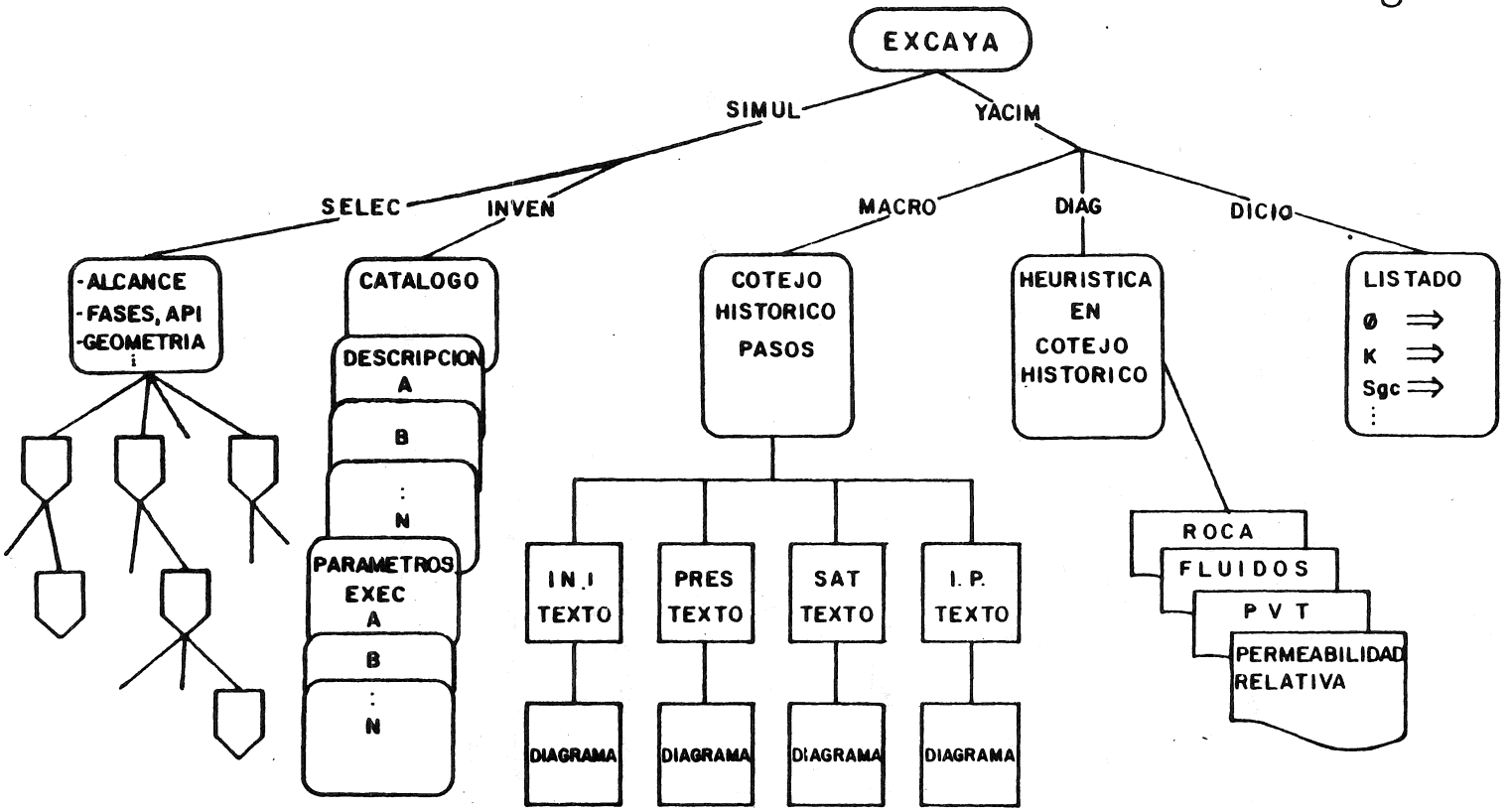
LAGOVEN

DEPARTAMENTO DE PRODUCCION

IP / IF

ANEXO

EXCAYA: AYUDA EXPERTA EN CARACTERIZACION DE YACIMIENTOS



TÉCNICAS AVANZADAS PARA LA GENERACION DE IMAGENES

CARLOS ENRIQUE ALVAREZ GUERRERO

Facultad de Ingenieria, Dpto. de Sistemas, Grupo de Investigación DFAC.

Universidad de Los Andes, Bogotá D.E., Colombia.

e-mail: DFAC@ANDESCOL.BITNET AA: 4976 Télex: 42343 Unand Co

Fax: 57(1)2841890 Tel: 57(1)28666185

Resumen - Se discuten los principales algoritmos para implementar un generador de imágenes por rastreo de rayos. Primero se analizan los algoritmos básicos de la técnica de rastreo. Posteriormente se presentan los métodos acelerados de intersección rayo-objetos y el algoritmo de generación de imágenes por duplicaciones sucesivas. Los objetivos son mostrar diversos métodos para implementar un algoritmo eficiente de rastreo de rayos, determinar los problemas que deben ser resueltos y proponer futuras líneas de trabajo en el área.

Palabras Claves: Geometría Constructiva de Sólidos, Caras de un sólido, Imágenes, Rastreo de Rayos, Inclusión en Sólidos, Envolventes Convexos, Complejidad de Algoritmos .

INTRODUCCION

Las técnicas avanzadas de generación de imágenes de objetos tridimensionales como la de rastreo de rayos supera en calidad a las imágenes generadas con técnicas de estructura de alambre, pero también involucran un orden de complejidad mucho mayor en el número de operaciones requeridas.

En los modeladores basados en el esquema de representación por Geometría Constructiva de Sólidos (GCS) partiendo de superficies cuadráticas, es muy costoso y complejo manejar la representación de la frontera. En este caso se aplica la técnica de rastreo de rayos, con su relativo alto costo en tiempo debido al número de operaciones involucradas.

Por esta razón se inició la investigación de métodos para mejorar los tiempos de respuesta de los algoritmos basados en la técnica de rastreo de rayos, logrando, como se muestra a continuación, resultados positivos y abriendo nuevas posibilidades para trabajos futuros.

DESARROLLO DE LA INVESTIGACION

La investigación comienza con el estudio de las ideas principales que explota la técnica de rastreo de rayos. Esta técnica modela el fenómeno físico de la formación de imágenes en la retina del ojo, que consiste en un rayo de luz que viaja desde una fuente y tras sucesivos choques con objetos 'rebota' hasta llegar al ojo.

El fenómeno real ocurre en paralelo involucrando millones de rayos de luz, que parten de las distintas fuentes, de los cuales sólo una porción llega hasta la retina del ojo contribuyendo en la formación de la imagen. Para aplicar herramientas computacionales en la generación de imágenes se modela el fenómeno invirtiendo el orden causa-efecto. Primero se discretiza el área sobre la retina, donde se forma la imagen, con una matriz de puntos, y para cada punto se calcula el rayo como si partiera de la retina rebotando sucesivamente con los objetos hasta llegar a las fuentes de luz. El modelo mostrado para generar imágenes se conoce como técnica de rastreo de rayos (ray tracing).

El primer resultado que se obtiene del anterior análisis es que el número de operaciones de los algoritmos que implementan la técnica de rastreo de rayos resulta de la cantidad de puntos necesarios para formar la imagen y la complejidad para calcular el color de cada punto. Resultados empíricos muestran que para generar imágenes de mediana resolución es necesario calcular 300 filas por 300 columnas de puntos, lo que da un total de 90.000 puntos. Disminuir esta cantidad es sacrificar la calidad de la imagen generada, anulando la principal característica de la técnica de producir imágenes 'reales'.

Visto que el número de puntos requeridos para generar una imagen no se puede disminuir, a continuación se analiza la complejidad del cálculo del color de cada punto de la imagen, que es el segundo factor en la complejidad total de los algoritmos de generación de imágenes.

Para calcular el color de cada punto de la imagen se modelan varios fenómenos de la física óptica: reflexión, refracción y reflexión difusa. Cuando un rayo de luz choca contra un objeto, una porción se refleja si la superficie del objeto tiene brillo y otra porción se refracta atravesándolo si la superficie tiene transparencia. El fenómeno de reflexión difusa se produce por rayos provenientes de las fuentes de luz y que se reflejan en todas direcciones al chocar contra el objeto. Modelando estos tres fenómenos se calcula el color para cada punto de la imagen, construyendo un rayo inicial que parte de la imagen y calculando las contribuciones por reflexión, refracción y reflexión difusa en el punto en donde el rayo choca con el objeto. Esta técnica para calcular el color de cada punto (figuras 1 y 2) es la que propiamente se conoce como rastreo de rayos.

El algoritmo básico para rastrear los rayos consiste en encontrar el siguiente punto en donde el rayo choca con un objeto de la escena. Este algoritmo se resuelve en dos etapas, en la primera se calculan los puntos de intersección del rayo con todos los objetos que se están visualizando, y en la segunda se escoge el punto de intersección con el que realmente choca el rayo.

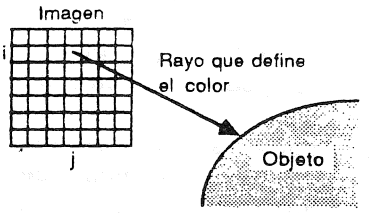


Fig 1. Cálculo del color de un punto de la imagen

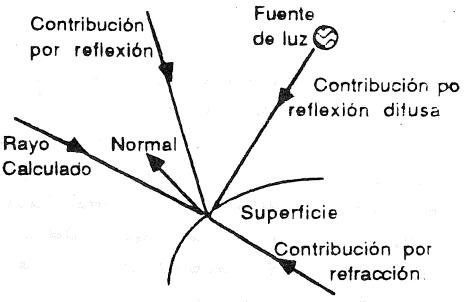


Fig 2. Cálculo del color de un rayo

En el esquema GCS no se mantiene la representación por la frontera de los sólidos, la intersección de un rayo con los objetos no se puede realizar a nivel de caras sino de superficies (entendiendo que una superficie es ilimitada mientras que una cara es una porción de superficie delimitada por contornos que determinan su frontera). Por esta razón el número de puntos de intersección de un rayo con los objetos es directamente proporcional al número de superficies de los objetos (figuras 3 y 4) incrementando la complejidad en la escogencia del verdadero punto de choque.

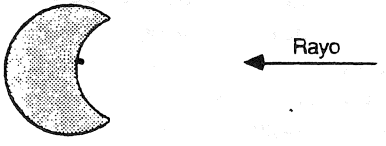


Fig 3. Intersección rayo-objeto

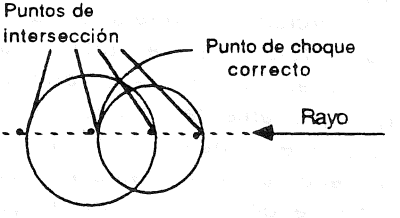


Fig 4. Intersección rayo-superficies

El algoritmo de escogencia del punto de choque correcto se realiza en dos pasos: primero se ordenan los puntos de intersección de menor a mayor con respecto a su distancia al origen del rayo, después se escoge el punto de choque como el primer punto que pertenece a la frontera del objeto.

Los algoritmos de intersección (rayo-objeto) e inclusión (para evaluar si un punto pertenece a la frontera de un objeto) se

mejoran para disminuir sus tiempos de respuesta, pero el tiempo total de generación de imágenes no disminuye considerablemente debido a que la complejidad en el número de operaciones es la misma.

Una mejora se realiza introduciendo envolventes convexos para delimitar el espacio ocupado por lo objetos, los cuales se definen a nivel de las primitivas del esquema de representación y se operan a medida que se realizan las operaciones de construcción. Los envolventes sirven como criterio para decidir si una evaluación sobre un objeto es aplicable o no sobre sus componentes.

El criterio para intersectar un rayo con un sólido es el siguiente: si el rayo NO intersecta el envolvente convexo del sólido entonces NO intersecta el sólido; de lo contrario, el cálculo recurre sobre los sólidos hijo-izquierdo e hijo-derecho. El criterio se aplica recursivamente hasta llegar a las primitivas en donde se calcula la intersección entre el rayo y las superficies.

La mejora consiste en detectar pronta y económicamente los casos en que el rayo no intersecta los objetos, ahorrando el cálculo infructuoso de intersecciones entre el rayo y todas las primitivas (superficies) de los objetos.

Los mismos envolventes se utilizan en los algoritmos de inclusión con el siguiente criterio: si un punto está por fuera del envolvente convexo de un sólido entonces está por fuera del sólido; en caso contrario, aplique el algoritmo de inclusión sobre los sólidos hijo-izquierdo e hijo-derecho, y determine la respuesta según la operación de construcción entre estos.

En el caso de la inclusión los envolventes permiten detectar rápidamente los casos en que los puntos están por fuera de los objetos, evitando calcular la inclusión con todas las primitivas que los componen.

Las estructuras de los envolventes convexos son útiles para detectar los casos en que NO es necesario recurrir en la evaluación de un objeto, pero no modifica la forma en que finalmente se realiza. En particular, no se obtiene ningún ahorro

en la evaluación cuando el criterio "no recurra" no es aplicable, y en cambio se recarga el número de operaciones por evaluar dicho criterio a los envolventes. Esto se tomó en cuenta para escoger a la esfera como envolvente convexo, que aunque no es el envolvente más pequeño para algunos casos, si tiene la operación más sencilla para evaluar el criterio.

CONTINUIDAD DE LA FRONTERA DE UN OBJETO

El aporte importante a la disminución de la complejidad de la generación de imágenes se realizó modificando el algoritmo general para calcular el color de cada punto de la imagen. Este aporte resultó de observar que la frontera de los objetos cambian con relativa suavidad, es decir, mantiene continua su derivada con muy pocos puntos de discontinuidad en donde se intersectan sus caras (aristas).

Al mirar la foto de un objeto, se encuentra que el porcentaje de los puntos que caen en las aristas es mucho menor que los que caen totalmente dentro de alguna cara. Visto de otra forma, se encuentra que el número de puntos que conforman las aristas de una cara es casi insignificante comparado con el número de puntos que conforman su interior.

Como cada cara del sólido es generada por SOLO UNA superficie, el párrafo anterior implica que las mayores áreas de las imágenes se pueden generar intersectando los rayos con una sola superficie de los objetos. Más aún, en los puntos sobre las aristas basta con intersectar el rayo con las caras vecinas a la arista, que de todos modos es un número pequeño de caras, generalmente constante.

La mejora introducida consiste en que para calcular el choque de un rayo con los objetos visualizables sólo se intersecta el rayo con UNA superficie de un objeto, generando únicamente dos puntos de intersección. El algoritmo de selección del punto de choque correcto también cambia: dado que sólo uno de los dos puntos de intersección tiene su normal dirigida hacia el observador, pues pertenecen a la misma superficie, el punto que cumpla esta condición es el punto de choque correcto. En total se obtiene que para calcular el choque de un rayo con los objetos visualizables

se realiza un número constante de operaciones que es realmente el MINIMO número posible, ya que de otra forma no se pueden modelar los fenómenos físicos de reflexión, refracción y reflexión difusa.

La nueva estrategia para calcular el punto de choque de un rayo con los objetos se basa en poder determinar de antemano si el rayo se dirige al interior de una cara y conocer la superficie que genera esa cara. Esta estrategia se puede implementar fácilmente si se tiene información acerca del punto de choque de los rayos vecinos. Suponiendo la configuración de la figura 5, donde cada celda contiene la información de la superficie que generó el punto de choque correcto para el respectivo rayo, la superficie de choque para el rayo de la mitad representado con '?' se puede calcular así: si las ocho celdas vecinas indican que los respectivos rayos chocaron con la misma superficie, S1 = ... = S8, se supone que el rayo se dirije al interior de una cara y se escoge la superficie S1 para calcular el punto de choque de ?; en caso contrario se escogen todas las superficies S1 hasta S8 como candidatas para generar el punto de choque de ?.

S1	S2	S3
S4	?	S5
S6	S7	S8

Fig 5. Información de Vecindad

En el párrafo anterior se muestran los dos casos mencionados al comienzo de esta sección: cuando un rayo choca al interior de una cara, o sobre la intersección de varias caras (aristas o vértices). Cuando las superficies S1 hasta S8 son la misma, el rayo de la mitad se dirige al interior de la cara generada por la superficie S1. En caso de que algunas de las superficies Si sean diferentes, probablemente el rayo se dirige a la intersección de varias caras y el punto de choque se calcula intersectando el rayo con las ocho superficies S1 hasta S8, que en el peor caso serán ocho superficies diferentes.

PROBLEMAS DE LA INFORMACION DE VECINDAD

En cualquiera de los dos casos mencionados, la **suposición** de encontrarse en el interior de una cara o en la intersección de máximo ocho caras puede fallar ocasionalmente cuando sobre el punto en el que choca un rayo con el objeto exista un accidente geométrico no detectado por los rayos vecinos. Un accidente así puede ser una superficie nueva muy pequeña que sobresale o un hueco dentro del objeto cuyo diámetro sea más pequeño que la distancia entre los puntos de choque de los rayos vecinos.

En las figuras 6 y 7, la superficie Sup2 es la base de una depresión del objeto ubicada sobre la superficie Sup1. La suposición de vecindad falla porque los rayos vecinos de R2, R1 y R3, chocan ambos contra la superficie Sup1.

Este inconveniente es fácilmente ignorado si se acepta que la probabilidad de que ocurra un accidente geométrico es muy pequeño y el error que se comete afecta sólo a un punto entre sus ocho vecinos. Lo anterior quiere decir que el error cometido sólo afecta ocasional y localmente puntos dispersos sobre la imagen, lo que no disminuye considerablemente su calidad, y en cambio la complejidad de los algoritmos se hace constante.

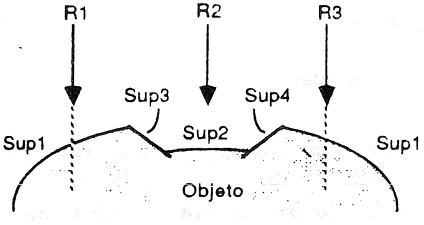


Fig 6. Accidente Geométrico sobre Sup1

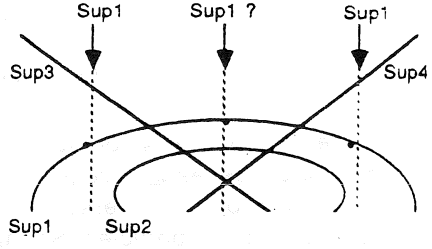


Fig 7. Error de la suposición de vecindad

Lo anterior es cierto, pero pierde sustentación cuando se implementa eficientemente la nueva estrategia. Para acelerar el cálculo de los puntos de choque de los rayos se tiene que disponer apriori de la información de los rayos vecinos. Esta información tiene que ser calculada en algún momento utilizando los algoritmos pesados de rastreo de rayos. Una buena

implementación debe garantizar que el número de rayos iniciales sea tan pequeño como sea posible y estén lo más uniformemente repartidos sobre la imagen.

El algoritmo utilizado para implementar la técnica de rastreo de rayos utilizando la información de vecindad trabaja por duplicación de imágenes. La idea es contruir las imágenes por duplicaciones sucesivas comenzando con otras tan pequeñas como sea posible y aumentándolas hasta conseguir el tamaño requerido. El nuevo esquema de trabajo explota los algoritmos acelerados en cada etapa de duplicación de la imagen, pero introduce fuertemente los problemas causados por los supuestos sobre la información de vecindad.

En las dos siguientes secciones se describen el algoritmo de duplicación de imágenes y se presentan soluciones al problema de la infomación de vecindad.

DUPLICACION DE IMAGENES

El problema de determinar apriori las superficies de choque de los rayos de una vecindad se puede definir recursivamente haciendo que estos rayos se calculen a partir de otro vecindario, hasta terminar finalmente en un conjunto de rayos que se calculan con los algoritmos pesados de rastreo de rayos. El objetivo de esta sección es definir un algoritmo para duplicar el tamaño de una imagen utilizando la información de vecindad y los algoritmos acelerados, para calcular los nuevos puntos de la imagen.

Para cumplir los anteriores objetivos se desarrollaron los siguientes pasos. Suponga una imagen de 3 filas y 3 columnas (figura 8). Si se conoce las superficie con la que choca el rayo correspondiente a cada punto de la imagen (por cada punto una superficie), es posible duplicar el tamaño de la imagen utilizando los puntos conocidos como vecindad de los nuevos puntos intermedios. La estrategia se muestra en las siguientes figuras en donde los asteriscos '*' indican puntos conocidos y los interrogantes '?' los puntos que se calculan.

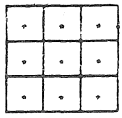


Fig 8. Imagen Original (3x3)

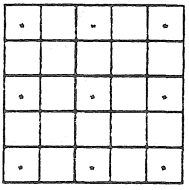


Fig 9. Imagen duplicada imcompleta

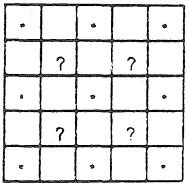


Fig 10. Primer paso

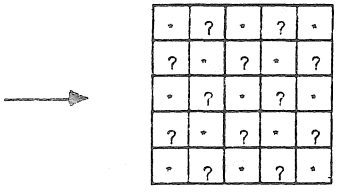


Fig 11. Segundo paso

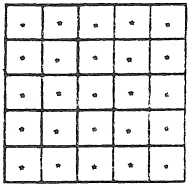


Fig 12. Imagen duplicada completa (5x5

Primero que todo se duplica el tamaño de la imagen con lo que quedan puntos intermedios indeterminados (celdas en blanco). Estos puntos se calculan utilizando los puntos conocidos como 'vecindad' en la forma mostrada en las figuras 10 y 11. En el primer paso, figura 10, se calculan los puntos sobre las filas y columnas pares de la imagen, utilizando los puntos vecinos sobre las filas y columnas impares (vecino superior-izquierdo, superior-derecho, inferior-izquierdo e inferior-derecho). En el segundo paso, figura 11, se calculan los puntos restantes utilizando los puntos vecinos en cruz (vecino superior, izquierdo, derecho e inferior).

Este es el proceso completo para duplicar el tamaño de una imagen. De esta forma es posible generar imágenes de cualquier tamaño comenzando con unas más pequeñas y repetiendo el proceso de duplicación tantas veces como sea necesario. Queda también claro como se utilizan los algoritmos acelerados que se basan en la información de vecindad para calcular los puntos nuevos de cada duplicación.

El problema debido a los supuestos por la información de vecindad fué mencionado a causa de la ocurrencia de accidentes

geométricos. Cuando las imágenes se generan por duplicaciones sucesivas, el efecto visual después de cada duplicación es el aumento en la resolución de la imagen . El algoritmo puede explotarse al máximo comenzando con imágenes rectangulares pequeñas, de por ejemplo 16 a 25 puntos iniciales, utilizando al mínimo los algoritmos pesados de cálculo de choque de rayos.

Cuando el número de puntos de la imagen inicial es muy pequeño, el nivel de resolución de la imagen es muy baja y todas las caras de los objetos se vuelven accidentes geométricos. Los errores cometidos se acumulan por las duplicaciones sucesivas produciendo el ocultamiento de caras. La manera más obvia de solucionar este problema es comenzar con imágenes tan sólo dos o tres veces más pequeñas que el tamaño requerido final, con el fin de disminuir la probabilidad de encontrar accidentes geométricos y localizar los posibles errores en áreas pequeñas de la imagen que no sean significativas en comparación con la imagen total.

Esta solución disminuye la probabilidad de cometer errores por la suposición de vecindad, pero no los evita. De otra parte incrementa el número de puntos que se calculan con los algoritmos pesados de cálculo de choque de rayos. Las soluciones que se proponen a continuación no evitan que se presente el problema de los accidentes geométricos, sino que los detecta. De esta forma se refuerza la suposición de vecindad con criterios más eficaces que determinan cuando es o no correcta la suposición.

SOLUCIONES AL PROBLEMA DE LA SUPOSICION DE VECINDAD

La suposición de vecindad establece que en la vecindad de un rayo se encuentra toda la información necesaria para calcular la superficie con la cual choca el rayo. Los errores debidos a esta suposición ocurren cuando se presentan accidentes geométricos, figuras 6 y 7. Para reforzar la suposición de vecindad se presenta una idea sencilla pero útil para detectar los accidentes geométricos.

Los accidentes geométricos son huecos en las caras de los objetos donde sobresale o se descubre otra cara. Estos huecos tienen un tamaño tal que aunque un rayo se dirija hacia él, ninguno de los rayos vecinos lo hace. Esta situación corresponde a una discontinuidad en la cara de un objeto que se 'oculta' por la suposición de vecindad. Si tal discontinuidad ocurre, entonces ninguno de los puntos de intersección calculados, entre el rayo y las superficies chocadas por los respectivos rayos vecinos, se localizan sobre la frontera del objeto (figuras 6 y 7). Esta condición sencilla pero importante se evalúa con los algoritmos de inclusión.

Con la anterior propuesta se modifica el algoritmo general de generación de imágenes por rastreo de rayos quedando de la siguiente forma:

1. Genere una imagen inicial con los algoritmos pesados de cálculo de choque de rayos:
   a. Genere la matriz de puntos que conforman la imagen inicial.
   b. Para cada punto de la matriz genere el rayo respectivo (el rayo que proviene de la escena y determina el color)
   c. Calcule el punto de choque (determinando la respectiva superficie) de cada rayo:
      i. Calcule los puntos de intersección con el algoritmo de intersección de rayo con todas las superficies de los objetos
      ii. Ordene los puntos de menor a mayor de acuerdo a su distancia al origen del rayo
      iii. Seleccione el punto de choque correcto evaluando el algoritmo de inclusión sobre los puntos

2. Duplique sucesivamente la imagen hasta obtener el tamaño requerido. Para cada duplicación:
   a. Duplique la matriz de la imagen.
   b. Para calcular los puntos nuevos siga los pasos mostrados en las figuras 9 a 12.
   c. Para calcular el punto de choque del rayo respectivo a cada punto:
      i. Calcule los puntos de intersección entre el rayo y las superficies de choque de los rayos vecinos.

ii. Ordene los puntos de menor a mayor de acuerdo a su distancia al origen del rayo.
iii. Seleccione el punto de choque correcto evaluando el algoritmo de inclusión sobre los puntos.
iv. Si ningún punto se localiza sobre la frontera de los objetos, recalcule el punto de choque utilizando los pasos del numeral c del primer punto (1.c).

RESULTADOS OBTENIDOS

Se implementaron todos los algoritmos y las mejoras propuestas y se realizaron pruebas de generación de imágenes con diferentes objetos. Se utilizó el esquema de representación GCS con envolventes convexos esféricos para mejorar los algoritmos de intersección e inclusión.

Se modelaron cinco objetos (sólidos tri-dimensionales) diferentes, nombrados y descritos en la tabla-1. Las primeras cuatro columnas describen el número de **superficies** planas, esféricas, cilíndricas y cónicas utilizadas para representar cada objeto, sumando el total en la quinta columna. Las últimas cinco describen el número de **caras** siguiendo la misma clasificación.

Objeto	s_pln	s_esf	s_cil	s_con	t_sup	c_pln	c_esf	c_cil	c_con	t_car
ansi	62	-	7	-	69	15	-	7	-	22
botella	20	2	4	2	28	2	2	4	2	10
decaucho	32	-	4	-	36	8	-	8	-	16
dif_cilin	8	4	2	-	14	-	2	4	-	6
empaque	86	-	22	-	108	5	-	24	-	29

Tabla 1. Descripción de los objetos modelados

El algoritmo general produce imágenes de rastreo de rayos con un nivel de exploración: cada rayo parte de la imágen hasta chocar contra un objeto de la escena en donde se calcula el color modelando el fenómeno de reflexión difusa. Primero se construye una imagen inicial utilizando el algoritmo original de **rastreo de rayos** (pasos mostrados en el numeral uno de la sección anterior). Para duplicar las imágenes se utilizaron tres tipos de algoritmos basados en las técnicas propuestas. El primer algoritmo, **vecindad-inclusión**, implementa los pasos mostrados en el

numeral dos de la sección precedente. El segundo, **vecindad**, difiere del anterior en que suprime el paso 2.c.iv. El último algoritmo, **interpolación**, utiliza directamente la información de color de los puntos vecinos para interpolar linealmente.

Para cada objeto modelado se generaron doce imágenes de 321 filas por 321 columnas, empleando diferentes combinaciones de algoritmos durante las duplicaciones. Como es de esperarse, los algoritmos más exactos se utilizaron en los primeros niveles de duplicación y los más rápidos en los últimos. Siguiendo este orden, se utilizó siempre primero el algoritmo de vecindad-inclusión, luego el de vecindad y por último el de interpolación. En la tabla 2 se resumen los tiempos y calidades de las imágenes obtenidas de cada objeto y cada tipo de combinación.

La primera columna de la tabla 2, **código**, indica la combinación de algoritmos utilizada en los diferentes niveles de duplicación de la imagen. El código se compone de tres dígitos, que leidos de izquierda a derecha indican el número de veces que se empleó el algoritmo de vecindad-inclusión, de vecindad, y de interpolación respectivamente. Para un tamaño de 321 puntos de lado, la imagen puede ser generada hasta por seis niveles de duplicación, siendo esta cantidad la suma máxima de los dígitos del código. El código '000', primera fila, indica que no se usó ningún nivel de duplicación, en otras palabras, toda la imagen fué generada inicialmente del tamaño total utilizando el algoritmo original de rastreo de rayos.

Para cada objeto, según la combinación utilizada, se describe el tiempo total de generación de la imagen, el porcentaje comparativo de tiempo con la imagen ideal (correpondiente a la combinación de la primera fila) y la calidad visual. Para todos los objetos se obtuvieron máximo tres niveles distintos de calidad de imagen calificadas de uno a tres, uno para la imagen "perfecta" y tres para la más errada.

En el anexo A se muestran los tipos de imágenes obtenidos, identificadas en correspondencia con la tabla 3.

imágen	ansi			botella			decaucho			empaque			dif_cilind		
Código	Tiemp	%	Cl	Tiemp	%	Cl	Tiemp	%	Cl	Tiemp	%	Cl	Tiemp	%	Cl
000	1163	100	1	550	100	1	1975	100	1	1651	100	1	297	100	1
600	651	60	3	339	62	3	1146	58	1	1358	82	1	272	92	1
060	95	8.2	3	55	10	3	161	8.2	1	107	6.5	3	49	16	1
003	93	7.9	2	51	9.3	1	144	7.3	1	189	11	1	36	12	2
222	83	7.1	3	46	8.4	3	124	6.3	1	177	10	1	34	11	1
123	82	7.0	3	45	8.2	3	125	6.3	1	176	10	1	33	11	2
004	83	7.1	2	45	8.2	2	127	6.4	1	178	10	1	33	11	2
114	82	7.0	3	45	8.2	3	120	6.1	1	175	10	1	33	11	2
105	83	7.1	3	46	8.4	3	123	6.2	1	175	10	1	33	11	3
051	85	7.3	3	47	8.5	3	132	6.7	1	101	6.1	3	37	12	1
015	83	7.1	3	46	8.4	3	123	6.2	1	99	5.9	3	33	11	3
006	82	7.0	3	45	8.2	3	120	6.1	1	99	5.9	3	32	11	3

Tabla 2. Resultados en tiempo (segundos) y calidad de imagen

Objeto	Calidad	Objeto	Calidad	Objeto	Calidad
botella	3	botella	2	botella	1
ansi	3	ansi	2	ansi	1
decaucho	1	empaque	3	empaque	1
dif_cilin	3	dif_cilin	2	dif_cilin	1

Tabla 3. Identificación de la imágenes

CONCLUSIONES Y TRABAJOS FUTUROS

En la práctica se diseñaron algoritmos para implementar las técnicas presentadas en este artículo y se obtuvieron resultados muy alentadores en cuanto a los tiempos de respuesta de generación de imágenes.

Los mayores avances se logran combinando varios métodos de duplicación de imágenes, utilizando sólo en los primeros níveles de duplicación, cuando la resolución es muy baja, los criterios para detectar accidentes geométricos y utilizando en los últimos niveles de duplicación la suposición de vecindad.

Otras formas de jugar con los niveles de duplicación es utilizar métodos de interpolación lineal o de segundo grado para duplicar las imágenes utilizando tan sólo la información del color. Si estos métodos se incluyen en los últimos niveles de duplicación,

la imagen no sufre mayor deterioro en cuanto a calidad y en cambio se obtienen tiempos de respuesta más cortos.

Combinar diferentes métodos en los niveles de duplicación ha reportado resultados muy interesantes en cuanto a la calidad de las imágenes generadas y los tiempos de respuesta. En este sentido se juega con un problema ya muy conocido expresado en la siguiente ecuación:

calidad_imagen / tiempo_generación = constante.

Sin embargo la generación de imágenes por duplicaciones sucesivas tiene dos características importantes muy favorables:
1. Para duplicar una imagen, el número de puntos nuevos que se calculan
es tres veces mayor que todos los puntos ya calculados.
2. En ún solo nivel de duplicación, los errores que se cometen por ocultamiento de caras ocurren con poca frecuencia y están lo suficientemente dispersos de modo que no afectan significativamente la calidad de la imagen total duplicada.

Las dos características mencionadas abren muchas alternativas para aplicar diversos métodos en los diferentes niveles de duplicación de las imágenes. La estrategia global se orienta a usar métodos más efectivos pero menos eficientes en los primeros niveles de duplicacicón, y métodos menos efectivos pero mas eficientes en los últimos. La conclusión de las dos características mencionadas es, que la calidad de las imágenes generadas por duplicaciones sucesivas se define en los primeros niveles de duplicación, mientras que el tiempo de respuesta total se define en los últimos niveles.

Otra alternativa que se abre para disminuir el tiempo de respuesta de generación de imágenes, independiente de la decisión de los métodos que se usan en los diferentes niveles de duplicación, es dividir la imágen en sub-imágenes y usar procesamiento en paralelo. El problema de generar una imagen de un tamaño dado es perfectamente divisible en el problema de generar n sub-imágenes independientes, dividiendo el tiempo de respuesta por el factor n, donde n es el número de procesadores que pueden actuar en paralelo.

En el presente trabajo se muestran varios algoritmos para calcular los puntos de choque de los rayos con los objetos de una escena. Para los algoritmos acelerados, basados en la suposición de vecindad, se muestra igualmente que el criterio de inclusión los hace más efectivos porque detectan los accidentes geométricos. Existe sin embargo un caso patológico que el criterio de inclusión no detecta cuando se aplica la suposición de vecindad.

El caso patológico se describe con las figuras 13 y 14, donde R1, R2 y R3 son rayos y Sup1, Sup2, Sup3, Sup4 y Sup5 son superficies de un objeto de la escena. Si se está calculando la superficie de choque de R2, los rayos vecinos R1 y R3 sólo mencionan las superficies Sup1 y Sup5. El problema es que aunque R2 corta a Sup5, y el punto de intersección pertenece a la frontera del objeto, la verdadera superficie de choque de R2 es Sup2.

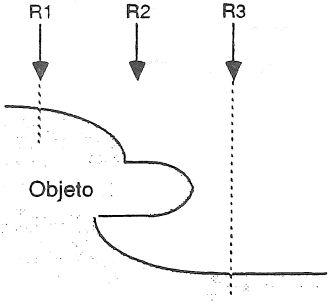


Fig 13. Caso Patológico de un accidente geométrico

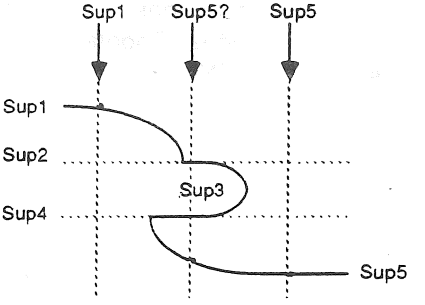


Fig 14. Error por la suposición de vecindad no detectado con inclusión

La anterior situación se muestra como un caso patológico que tiene que ser tratado y solucionado en trabajos posteriores. También se propone como trabajos futuros la corrección de imágenes, mediante barridos que detecten puntos con cambios bruscos en el color o que pertenezcan a superficies diferentes a las superficies de todos los puntos vecinos.

BIBLIOGRAFIA

[LEV76] LEVIN, J. A parametric algorithm for drawing pictures of solid objects composed of cuadric surfaces. Comm. ACM. 19, 10. (Oct 1976) Págs 555-563.

[MAN88] MANTILA, M. An introduction to Solid Modeling. Comput. Science Press, Inc. Rockville, Maryland. 1988.

[MIL88] MILLER, J. R. Analysis of Quadirc-Surface-Based Solid Models. IEEE, Comp. Graphics & Applications. January, 1988. págs 28-42.

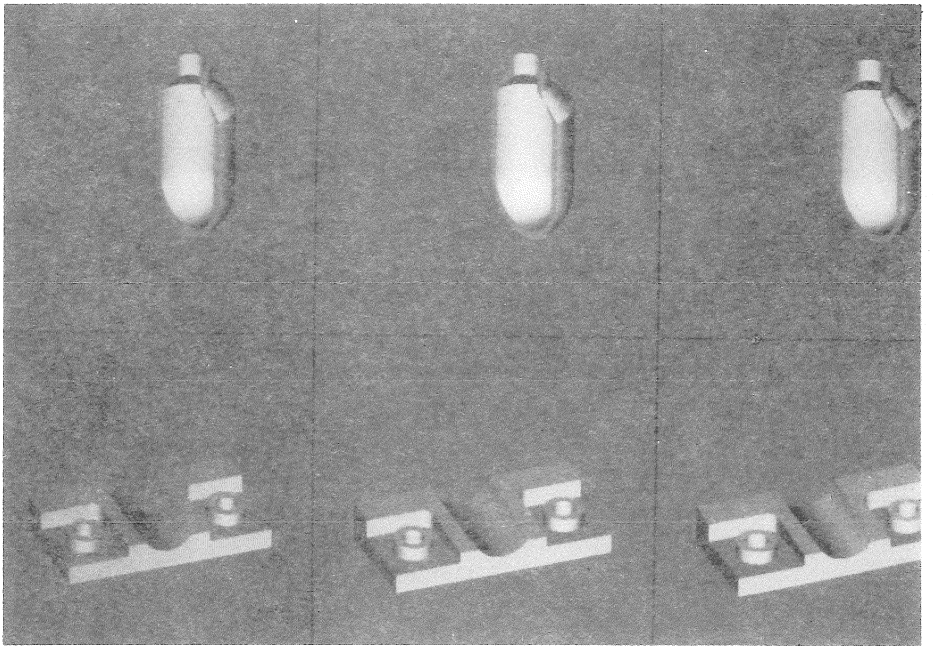
[YAN87] YANG, C-G. Illumination Models for generating images of curved surfaces. Computer-aided design. Vol 29, Nº 10. (Dec 1987) Butterworth & Co Ltda. págs 544-554.

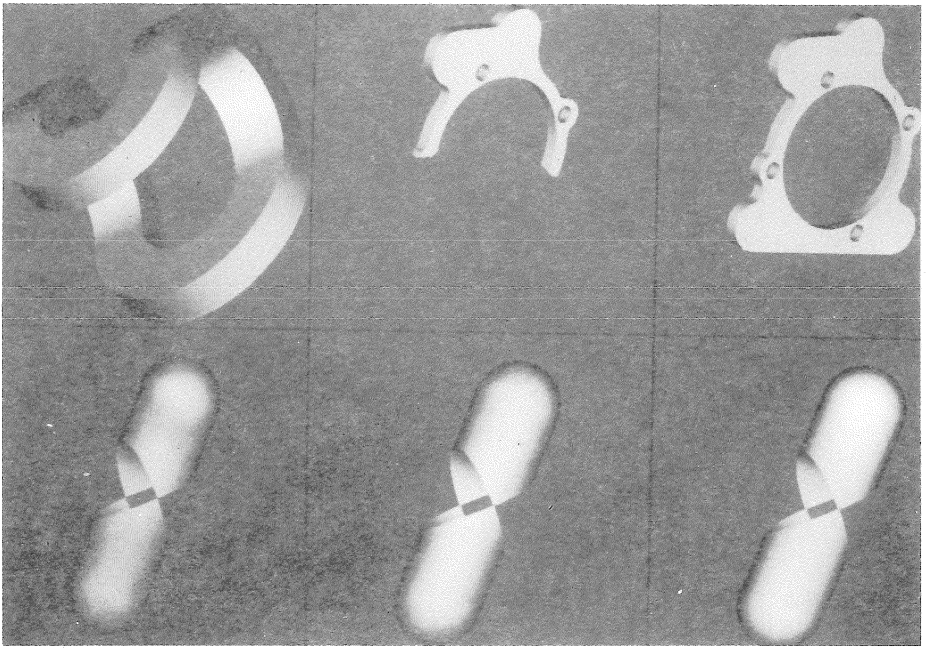
[HEC87] HECKBERT, P. S. Writing a Ray Tracer. Introduction to Ray Tracing Tutorial Notes. SIGGRAPH'87, July 1987.

[ALV90] ALVAREZ, C. Superficies Cuadráticas para el Modelaje de Sólidos. Tésis de Grado, CIFI, Universidad de Los Andes, Bogotá DE, Colombia. 1990.

[HAI] HAINES, E. Essential Ray Tracing Algorithms.

Anexo A. Niveles de Calidad de las imágenes generadas





Un Algoritmo para el Problema de Asignación Generalizado con Múltiples Recursos.

Alejandro Crema.
Escuela de Computación.Facultad de Ciencias. U.C.V.
Apartado:47002 Caracas/Venezuela.
Fax:6627121 (Facultad de Ciencias/U.C.V.

RESUMEN

En este trabajo se presenta un Algoritmo de Ramificación y Acotación para el problema de Asignación Generalizado con Múltiples Recursos. La selección de los Problemas Candidato es hecha vía LIFO;se considera una cadena de Relajaciones:la Lagrangiana conducente a un conjunto de problemas independientes tipo Mochila con Múltiples Recursos,la resultante de considerar únicamente los recursos peor evaluados en términos relativos y su correspondiente Relajación Lineal y,finalmente,la resultante de eliminar las restricciones de Asignación.

Un conjunto de Lemas asociados a los valores optimales de las Relajaciones y sus relaciones es demostrado.

El Algoritmo sugerido explota la cadena de Relajaciones a los efectos de la clausura de los Problemas Candidato.La implantación actual se restringe a las dos últimas Relajaciones consideradas.

Se presentan resultados computacionales satisfactorios,en microcomputadores personales,para problemas asociados a una aplicación con un máximo de 1000 variables binarias y 15 restricciones.

I.- Introducción:

El objetivo fundamental de los problemas clásicos de Asignación y sus extensiones es encontrar apareamientos óptimos de "Agentes" y "Tareas" de tal manera de minimizar un funcional de costos.En [11] y [12] los autores presentan un conjunto de problemas que pueden ser tratados en tal contexto.

Un común denominador de tales problemas es la presencia de cotas para la disponibilidad de un "Recurso".La asignación de una Tarea a un Agente trae como consecuencia el consumo de una predeterminada cantidad del Recurso y la asignación global debe respetar la disponibilidad asociada a cada uno de los Agentes. La extensión del problema al caso de Múltiples Recursos puede verse en [2],[4] y [12].

En este trabajo se presenta un Algoritmo de Ramificación y Acotación (R. y A.) para el Problema de Asignación Generalizado con Múltiples Recursos.En la sección II puede verse la formulación Matemática del problema.El Algoritmo es presentado en la sección III y en IV pueden verse los Resultados y Conclusiones.

II.- El Problema de Asignación Generalizado con Múltiples Recursos y su Formulación Matemática:

M Tareas han de ser asignadas a N Agentes de tal manera que se minimîce el Costo Total a la vez que se respeten cotas en la disponibilidad de L Recursos asociados a cada uno de los Agentes.

Sean I=(1,...,N) , J=(1,...,M) y K=(1,...L).

Se supone conocido:el Costo por asignar la j-ésima Tarea al i-ésimo Agente:c(i,j), con i en I y j en J;el Consumo de unidades

del Recurso del tipo-k que debe hacer el i-ésimo Agente para realizar la j-ésima Tarea:r(i,j,k), con i en I, j en J y k en K; la disponibilidad del recurso del tipo-k asociada al i-ésimo Agente: R(i,k) , con i en I y k en K.Todos los parámetros involucrados en el problema se asumen positivos (con R(i,k) estrictamente positivo) y cada Tarea debe ser asignada a un único Agente.

A continuación se presenta la formulación matemática del problema al cual nos referiremos como P en lo que resta del trabajo.Las variables de decisión binarias, X(i,j) ,denotan si la j-ésima Tarea debe ser asignada al i-ésimo Agente (X(i,j)=1) o no (X(i,j)=0).En (1) puede verse el funcional de Costo Total.En (2) observamos las N x L restricciones asociadas a la disponibilidad de los Recursos.Las restricciones clásicas de Asignación aparecen en (3).

$$(1)\ MIN \sum_{i=1}^{N} c(i,j) \times X(i,j) \quad s.a:$$

$$(2)\ \sum_{j=1}^{M} r(i,j,k) \times X(i,j) \leq R(i,k) \quad \forall i \in I\ ,\ \forall k \in K$$

$$(3)\ \sum_{i=1}^{N} X(i,j) = 1 \quad \forall j \in J$$

$$(4)\ X(i,j) \in \{0,1\} \quad \forall i \in I\ ,\ \forall j \in J$$

Si hacemos L=1 en P,estaremos en presencia del Problema de Asignación Generalizado (General Assingment Problem,- GAP -, en

la lit. del área) para el cual existen algoritmos eficientes ([1] y [10]).

III.- El Algoritmo:

III.1.- Marco Conceptual:

P es un problema de Programación Entera-Lineal-0-1. La estrategia de solución que se presenta es la de R. y A..Los conceptos fundamentales y la terminología utilizada en relación a algoritmos basados en R. y A. pueden verse en el trabajo ,fundamental en el área,de Geoffrion y Marsten ([6]).

III.1.1.- Los Problemas Candidato (PC) :

A los efectos del presente estudio un Problema-Candidato, PC ,resulta de añadir a P restricciones del tipo: X(i,j)=1 ó X(i,j)=0 ,para pares (i,j) seleccionados.

En otras palabras alguna(s) Tarea(s) es(son) asignada(s) a algún(os) Agente(s) o bien algún(os) Agente(s) es(son) excluídos del conjunto que puede realizar determinada(s) Tarea(s).

III.1.2.- Relajaciones de P:

III.1.2.1.- Relajación Básica:

La Relajación Básica consiste en excluir de P el bloque de restricciones tipo (2), asociadas a la disponibilidad de los Recursos ,obteniéndose un problema clásico de Asignación ([8]) cuya solución es trivial.

En lo que sigue nos referiremos a tal Relajación como a PASIG ó PC-ASIG.

III.1.2.2.- Relajaciones sucesivas tipo Mochila:

(i) Considérese la Relajación Lagrangiana de P con respecto al bloque de restricciones (3), asociadas a la obligatoriedad de asignar cada Tarea a un único Agente,como sigue:

$$(1)' \quad MIN \sum_{i=1}^{N}\sum_{j=1}^{M} c(i,j) \times X(i,j) + \sum_{j=1}^{M} y(j) \times (1 - \sum_{i=1}^{N} X(i,j)) \quad s.a:$$

$$(2) \quad \sum_{j=1}^{M} r(i,j,k) \times X(i,j) \leq R(i,k) \quad \forall i \in I \ , \ \forall k \in K$$

$$(4) \quad X(i,j) \in \{0,1\} \quad \forall i \in I \ , \ \forall j \in J$$

En (1)' puede observarse la incorporación a (1) del bloque de restricciones (3) ponderadas según los multiplicadores y(j),para j=1,...,M.Sean i1(j), con j=1,...M ,tales que X(i1(j),j) = 1 en una solución óptima de PASIG y sean i2(j) tales que:

$$c(i2(j),j) \leq c(i,j) \quad \forall i \in I \ t.q \ i \neq i1(j) \ , \ \forall j \in J$$

En [10],Ross y Soland sugieren el uso de y(j)=c(i2(j),j) para el caso de asignación con un solo recurso.En este trabajo se asumen los mismos multiplicadores.Obsérvese que i2(j) no es más que el Agente que realiza la j-ésima Tarea al mínimo costo con la excepción ,posiblemente, de i1(j).Así, la relajación Lagrangiana,que denotaremos P(y), puede escribirse como sigue:

$$(1)' \sum_{j=1}^{M} c(i2(j),j) - MAX \sum_{j=1}^{N} \sum_{j=1}^{M} (c(i2(j),j) - c(i,j)) \times X(i,j) \quad s.a:$$

$$(2) \quad \sum_{j=1}^{M} r(i,j,k) \times X(i,j) \leq R(i,k) \quad \forall i \in I \ , \ \forall k \in K$$

$$(4) \quad X(i,j) \in \{0,1\} \quad \forall i \in I \ , \ \forall j \in J$$

Tal interpretación permite separar P(y) en N subproblemas independientes tipo Mochila con múltiples Recursos (uno para cada uno de los Agentes).Para una discusión extensa sobre los problemas tipo Mochila con múltiples recursos puede verse [4] y [13].

Puesto que es útil a los efectos de la presentación del Algoritmo,supongamos que hemos resuelto PASIG. Excepto para problemas triviales alguna(s) de las restricciones en el bloque (2) resultará(n) violada(s) por la solución obtenida para PASIG.Sean entonces:

$$J(i) = \{j \in J \ \ t.q: \ \ i1(j) = i\} \ , \ \forall i \in I$$

$$V(i) = \{k \in K \ \ t.q: \ \sum_{j \in J(i)} r(i,j,k) > R(i,k)\} \quad \forall i \in I$$

$$IINF = \{i \in I \ t.q: \ V(i) \neq \emptyset\}$$

$$EXC(i,k) = \sum_{j \in J(i)} r(i,j,k) - R(i,k) \quad \forall i \in IINF \ y \ k \in V(i)$$

Así, J(i) es el conjunto de las Tareas asignadas al i-ésimo Agente en la solución de PASIG;V(i) es el conjunto de recursos para los cuales se violó la disponibilidad asociada al i-ésimo Agente;IINF es el conjunto de Agentes para los cuales existe algún Recurso en situación de infactibilidad y EXC(i,k) es el exceso en el k-ésimo Recurso asignado al i-ésimo Agente.Alguna(s) de las Tareas asignada(s) a Agentes en IINF debe(n) ser reasignadas en la búsqueda de factibilidad.El mínimo incremento posible en tales casos ,para el funcional de Costos, viene dado por:

$$DIF(j)=c(i2(j),j) - c(i1(j),j)$$

Considérense los siguientes problemas independientes tipo Mochila con Múltiples recursos, uno para cada uno de los Agentes en IINF , que denotaremos M(i):

$$MIN \sum_{j \in J(i)} DIF(j) \times Y(i,j) \quad s.a:$$

$$\sum_{j \in J(i)} r(i,j,k) \times Y(i,j) \geq EXC(i,k) \quad \forall k \in V(i)$$

$$Y(i,j) \in \{0,1\} \quad \forall j \in J(i)$$

La interpretación de las variables binarias en M(i) es como sigue: Si Y(i,j) = 1 entonces la j-ésima Tarea debe ser reasignada en la búsqueda de factibilidad.

En el resto del trabajo si Q es un problema de optimización

entonces v(Q) denota el valor de una solución óptima de Q (si existe) y SF(Q) el conjunto de soluciones factibles.

Acerca de la relación entre v(P(y)),v(PASIG) y v(M(i)) para i en IINF, podemos demostrar el siguiente Lema-1 (ver prueba en Anexo):

$$v(P(y)) = v(PASIG) + \sum_{i \in IINF} v(M(i))$$

(ii) En la búsqueda de un compromiso entre la calidad de las Cotas y la eficiencia Computacional se relajó P(y) al considerar en (2) solo aquellas restricciones con k(i) escogido como sigue:

sean,

$$EXR(i,k) = EXC(i,k)/R(i,k) \quad \forall i \in IINF \ , \ \forall k \in V(i)$$

$$k(i) \ t.q. \ EXR(i,k(i)) = MAX_{k \in V(i)} \{EXR(i,k)\}$$

Sea PN(y) el problema resultante y MN(i) los problemas tipo Mochila asociados (con un solo recurso: k(i)).Puede entonces demostrarse el siguiente Lema-2 (ver prueba en Anexo):

$$v(PN(y)) = v(PASIG) + \sum_{i \in IINF} v(MN(i))$$

PN(y) es definido de tal manera que la búsqueda de factibilidad por reasignación de Tareas se concentra sobre

aquellos recursos en los cuales la violación de la disponibilidad es máxima en términos relativos.

Un último esfuerzo de conciliación Cotas-Eficiencia consiste en considerar la Relajación Lineal de PN(y), que denotaremos PNL(y) ,resultante de permitir:

$$0 \leq X(i,j) \leq 1 \quad \forall i \in I \ , \ \forall j \in J$$

En tal caso,al definir análogamente MNL(i),asumiendo la posibilidad de "reasignaciones parciales",con:

$$0 \leq Y(i,j) \leq 1 \quad \forall i \in IINF \ , \ \forall j \in J(i)$$

,se pueden demostrar los siguientes Lemas (ver prueba en anexo)

$$Lema\text{-}3 : v(PNL(y)) = v(PASIG) + \sum_{i \in IINF} v(MNL(y))$$

$$Lema\ 4 : v(PASIG) \leq v(PNL(y)) \leq v(PN(y)) \leq v(P(y)) \leq v(P)$$

III.1.3.- Clausura de PC:

III.1.3.1.- Por acotamiento y/o Factibilidad:

El Algoritmo que se presenta explota la cadena de Relajaciones en el sentido de que se intenta clausurar con PC-ASIG (alternativa asociada a la peor cota pero con la máxima eficiencia computacional) y si se fracasa se comienza a resolver, uno a uno ,los problemas tipo Mochila (MNL(i) ó MN(i) ó M(i)) con la esperanza de clausurar sin completar toda la cadena.Adicional-

mente y como es usual PC es clausurado si la solución de PC-ASIG es factible en P.

III.1.3.2.- Por Infactibilidad:

Se desarrollaron dos Tests de Infactibilidad, como sigue:

(i) Al evaluar PC se detecta que existe una restricción particular en situación de infactibilidad ,digamos asociada al par (i,k),que no puede llevarse a factibilidad porque un subconjunto de las Tareas asignadas al Agente-i,no reasignables como consecuencia de las asignaciones fijas realizadas durante el proceso,basta para implicar la infactibilidad.

(ii) Al evaluar PC se observa que existe una restricción particular, digamos la correspondiente a (i,k),que no puede llevarse a situación de factibilidad porque todas las Tareas asignadas al i-ésimo Agente en PC-ASIG y que son candidatas a reasignación,cumplen con:r(i,j,k)=0.

III.1.4.- Definición de las Variables de Ramificación y del Proceso de Ramificación:

En contraposición a los criterios usuales de Ramificación en los cuales, para el presente contexto, se seleccionaría una Tarea a ser definitivamente asignada a un Agente particular (ver el criterio de Ross y Soland,en [10],para un solo Recurso),se desarrolló un criterio de selección en el cual un Agente es excluído del conjunto al cual puede ser asignada una Tarea particular.

El criterio desarrollado es el siguiente:

(i) Seleccionar el par (i,k), digamos (IPEOR,KPEOR) cuya

restricción es la peor evaluada en términos relativos (en la solución de PASIG). Es decir:

$$EXR(IPEOR,KPEOR) = MAX_{i\in INF} EXR(i,k(i))$$

(ii) Seleccionar la Tarea j, digamos JSALE, con el menor cociente diferencial de costos-recurso,es decir:

$$DIF(JSALE)/r(IPEOR,JSALE,KPEOR) = MIN_{j\in JJ} DIF(j)/r(IPEOR,j,KPEOR)$$

$$con\ JJ = J(IPEOR) \cap \{ j\ t.q.: r(IPEOR,j,KPEOR) > 0 \}$$

Aquí cabe la siguiente observación: Si JJ es vacío entonces el problema bajo evaluación es infactible según el Test de infactibilidad (ii) de III.1.3.2.

(iii) La restricción x(IPEOR,JSALE)=0 es añadida a PC para definir el próximo problema a considerarse.El PC correspondiente a X(IPEOR,JSALE)=1 es añadido a la Lista de Problemas Candidato.

La idea que sustenta el criterio desarrollado es la siguiente:Se intenta "arreglar" la restricción peor evaluada en términos relativos. La Tarea JSALE es la "primera candidata" a ser reasignada pues posee el menor cociente diferencial-recurso. Tal criterio debe resultar eficiente en aquellos problemas en los cuales el porcentaje de reasignaciones de Tareas sea bajo y adicionalmente el porcentaje de reasignaciones al segundo agente mejor evaluado sea alto.En V podemos observar evidencia empírica de la eficiencia.

III.2.- Algoritmo de Ramificación y Acotación:

A continuación se presenta el Algoritmo de R. y A. desarrollado.A los efectos de simplificar la exposición se utilizan solamente las Relajaciones PASIG y PNL(y).La extensión al caso en el cual se explota toda la cadena de Relajaciones es inmediata.El estilo de presentación del Algoritmo ,ampliamente respetado en la literatura del área,es el sugerido por Geoffrion y Marsten ([6]).

ALGORITMO PROPUESTO

PASO-0:Iniciar ZMEJOR en un valor suficientemente grande.

PASO-1:

$$\forall j \in J \text{ y } \forall k \in K \text{ sea } imin(j,k) \text{ t.q: } r(imin(j,k),j,k) \leq r(i,j,k) \ \forall i \in I$$

$$Si \ \{k \text{ t.q: } \sum_{j=1}^{M} r(imin(j,k),j,k) > \sum_{i=1}^{N} R(i,k)\} \neq \emptyset$$

finaliza el proceso con mensaje de Infactibilidad.

PASO-2:Resolver PASIG.

PASO-3:Si la solución de PASIG es factible en P:

(i) Se actualiza ZMEJOR como sigue: ZMEJOR=v(PASIG).

(ii) Se asignan los valores de las variables de decisión como sigue: X(i1(j),j)=1 para cada j en J.

(iii) Finaliza el proceso.

PASO-4:Hacer PC = P.

PASO-5:Si PC resulta infactible segun Test (i) ir al PASO-15.

Si JJ es vacío (infactibilidad según (ii)) ir al PASO-15.

PASO-6:Se encuentra la Variable de Ramificación:(IPEOR,JSALE)

PASO-7:La restricción X(IPEOR,JSALE)=0 es añadida a PC definiendo el próximo problema a ser evaluado.El problema con X(IPEOR,JSALE)=1 ingresa a la LISTA.Se almacena el valor de la relajación asociada:v(PC-ASIG).

PASO-8:Se reasigna JSALE (equivale a resolver PC-ASIG).

PASO-9:Se intenta clausurar por acotación:

Si v(PC-ASIG) ≥ ZMEJOR PC es clausurado e ir PASO-15

PASO-10:Si la solución de PC-ASIG es factible en P:

(i) Se actualiza ZMEJOR como sigue: ZMEJOR=v(PC-ASIG).

(ii) Se actualiza el valor de las variables de decisión con los valores correspondientes a PC-ASIG.

(iii) Ir PASO-15 (PC es Clausurado).

Si no se conoce aún alguna solución factible ir al PASO-5.

PASO-11:Sea i en IINF.Se resuelve MNL(i) (asociado a PC).

PASO-12:v(PC-ASIG) := v(PC-ASIG) + v(MNL(i)).

PASO-13:Se intenta clausurar por acotación:

Si v(PC-ASIG) > ZMEJOR ir al PASO-15 (PC es clausurado)

PASO-14:IINF:=IINF-i.

Si Card(IINF)=0 ir al PASO-5 (Se fracasó en el intento de clausurar PC por acotación).

Si no ir PASO-11.

PASO-15: Si la Lista de Problemas Candidato esta vacía finaliza el proceso.En caso contrario se encuentra el nuevo PC por aplicación de la regla LIFO e ir PASO-11.

IV.-Resultados y Conclusiones:

El Algoritmo está implantado en FORTRAN sobre una PS-50.

Las relajaciones consideradas en la implantación son PASIG y PNL(y).

En las Tablas 1,2 y 3 se presenta un resumen de los resultados computacionales asociados a una aplicación con 5 Agentes y 3 recursos (para un total de 15 restricciones).La cantidad de Tareas se hizo variar en 100,150 y 200 (para un total de 500,750 y 1000 variables binarias).

En las Tablas,Radio representa el peor cociente entre el total de recursos requeridos por las Tareas y el total disponible por los Agentes.Para cada Radio en el intervalo [.4,.925] ,con un paso de 0.05,se resolvió un Lote de 10 problemas generados al azar respondiendo a la configuración particular de los datos de la aplicación:

- la disponibilidad relativa de los Agentes,para todo recurso,viene dada por:4/11,2/11,2/11,1/11,2/11.

- los requerimientos de las Tareas son generados al azar en el conjunto {0,1,2}.

- el costo asociado a la asignación de la j-ésima Tarea al i-ésimo Agente,es proporcional a los requerimientos globales de la Tarea y al tiempo de desplazamiento (que se supone proporcional a la distancia) desde la posición del Agente (en

puntos del Plano preestablecidos) hasta la de la Tarea (generada al azar).

El resto de la nomenclatura de las Tablas es la siguiente:Nodos es el promedio de PC evaluados;Reasig. es el promedio de Tareas reasignadas;TCPU es el promedio de tiempo de procesamiento (en seg.);SF-1 es el PC,en promedio, en el cual se encontró la primera solución factible;MaxTCPU es el peor tiempo de procesamiento para el lote de problemas y Reasig. a su derecha es la cantidad asociada de Tareas reasignadas.

Los tiempos de procesamiento se mantuvieron dentro de límites aceptables,según la experiencia con cerca de 7200 problemas,salvo para problemas con Radio superior ,aprox.,a .90 (resultados similares se encontraron para tres configuraciones distintas,variando la disponibilidad relativa de los Agentes con radio crítico en .7 para el peor de los casos).En tales casos los problemas fueron resueltos,con optimalidad,en una PS-80 con coprocesador matemático,en tiempos inferiores al límite de procesamiento,establecido en 600 seg.(se estima,conservadoramente,una relación 1/10 sobre los tiempos).Los resultados cualitativos (v(P) vs. Radio; Reasig. vs. Radio; TCPU vs. Radio; TCPU vs. M; TCPU. vs. Reasig.) son todos coherentes con lo esperado.

Algunas posibilidades para mejorar el Algoritmo implantado,no evaluadas hasta la fecha,son:

- Implantar toda la cadena de Relajaciones: (PASIG,PNL(y),PN(y) y P(y)) y ,adicionalmente,considerar distintas estrategias de solución (puede resolverse toda la

cadena para cada i en IINF ó resolver las Relajaciones en el orden sugerido por la cadena propuesta).Aquí cabe la siguiente observación:el esfuerzo computacional involucrado en la resolución de PNL(y) no está necesariamente perdido a los efectos de solucionar PN(y) y así sucesivamente.

- Considerar relajaciones adicionales:

Un problema tipo GAP al escoger un Recurso como el único activo y/o a partir de la definida por Jörnsten y Näsberg en [9] para el GAP y/o a partir de la relajación Mixta (Lagrangiana y Surrogate combinadas) que puede verse en [7],utilizada para P,con "pocas" restricciones (unas 5 según la referencia), por Gavish y Pirkul ([2]).

- Considerar reglas no-LIFO.

- Seleccionar los Agentes en IINF en un orden predeterminado según su potencial contribución al valor del funcional de la relajación de turno.

- Considerar otros criterios de ramificación,por ejemplo: el menor cociente diferencial-recurso tomado sobre todas la ternas (i,j,k) con i en IINF,j en J(i) y k en V(i) en lugar de restringir la escogencia a i=IPEOR y k=k(i).

- Considerar estrategias para la generación de soluciones factibles basadas en la información proveniente de las relajaciones.

Radio	Nodos	Reasig.	TCPU	SF-1	MaxTCPU	Reasig.
.50	4	2	1.28	2	2.36	5
.75	34	12	6.28	13	18.95	16
.80	254	18	52.50	22	120.89	22
>.90	2878	23	>600.00	36	-	-

M = 100 Restricciones: 15

TABLA-1

Radio	Nodos	Reasig.	TCPU	SF-1	MaxTCPU	Reasig.
.50	3	1	1.64	2	4.61	7
.75	47	14	13.53	16	31.47	17
.80	253	23	88.05	24	248.87	28
>.90	1931	27	>600.00	33	-	-

M = 150 Restricciones: 15

TABLA-2

Radio	Nodos	Reasig.	TCPU	SF-1	MaxTCPU	Reasig.
.50	8	4	4.09	5	10.44	12
.75	121	25	50.72	26	206.35	23
.80	603	27	283.82	35	571.77	32
>.90	1489	39	>600.00	43	-	-

M = 200 Restricciones: 15

TABLA-3

ANEXO

A continuación se presentan las pruebas de los Lemas introducidos en este trabajo:

$$Lema\text{-}1:\ v(P(y)) = v(PASIG) + \sum_{i \in IINF} v(M(i))$$

Prueba del Lema-1:

(1) De la definición de P(y) se sigue que:

$$Si\ i \notin IINF, podemos\ hacer\ X(i,j) = 1\ si\ i1(j) = i$$

$$Si\ i \in IINF, podemos\ hacer\ X(i,j) = 0\ si\ j \notin J(i)$$

(2) Consideremos los sub-problemas auxiliares Ma(i),para cada i en IINF,definidos como sigue:

$$MAX \sum_{j \in J(i)} (c(i2(j),j) - c(i1(j),j)) \times X(i,j)\ s.a:$$

$$\sum_{j \in J(i)} r(i,j,k) \times X(i,j) \leq R(i,k) \quad \forall\ k \in K$$

$$X(i,j) \in \{0,1\} \quad \forall j \in J(i)$$

(3) De (1) y (2) se sigue de inmediato que :

$$v(P(y)) = \sum_{j=1}^{M} c(i2(j),j) - \sum_{i \in IINF} \sum_{j \in J(i)} (c(i2(j),j) - c(i1(j),j))$$

$$- \sum_{i \in IINF} v(Ma(i))$$

(4) Consideremos el cambio de variable X(i,j)=1-Y(i,j) en cada Ma(i) y definamos el sub-problema auxiliar Ma2(i) para cada

i en IINF como sigue:

$$MAX \sum_{j \in J(i)} (c(i2(j),j) - c(i1(j),j)) \times (-Y(i,j)) \text{ s.a:}$$

$$\sum_{j \in J(i)} r(i,j,k) \times Y(i,j) \geq \sum_{j \in J(i)} r(i,j,k) - R(i,k) = EXC(i,k) \quad \forall k \in K$$

$$Y(i,j) \in \{0,1\} \quad \forall j \in J(i)$$

(5) De la definición de Ma2(i) se sigue que:

$$v(Ma(i)) = \sum_{j \in J(i)} (c(i2(j),j) - c(i1(j),j)) + v(Ma2(i)) \quad \forall i \in IINF$$

(6) De (3),(4) y (5) se sigue que:

$$v(\{P(y)\}) = \sum_{i \in IINF} \sum_{j \in J(i)} c(i1(j),j) + \sum_{i \in IINF} \sum_{j \in J(i)} c(i1(j),j) - \sum_{i \in IINF} v(Ma2(i))$$

(7) Claramente se tiene que:

$$v(PASIG) = \sum_{i \in IINF} \sum_{j \in J(i)} c(i1(j),j) + \sum_{i \in IINF} \sum_{j \in J(i)} c(i1(j),j)$$

(8) De las definiciones de M(i) y Ma2(i) se tiene que:

$$v(M(i)) = - v(Ma2(i)) \quad \forall i \in IINF$$

(9) De (6),(7) y (8) se concluye:

$$v(P(y)) = v(PASIG) + \sum_{i \in IINF} v(M(i))$$

$$Lema\text{-}2: \; v(PN(y)) = v(PASIG) + \sum_{i \in IINF} v(MN(i))$$

Prueba del Lema-2:

La prueba del Lema-2 es análoga a la del Lema-1 colocando

$k=k(i)$ *y no* $\forall k \in K$ *(con lo cual se redefinen* $Ma(i)$ *y* $Ma2(i)$ *)*

$PN(y)$ *en lugar de* $P(y)$ *y* $MN(i)$ *en lugar de* $M(i)$*.*

Lema-3: $v(PNL(y)) = v(PASIG) + \sum_{i \in TMF} v(MNL(i))$

Prueba del Lema-3:

La observación (1) en la prueba del Lema-1 sigue siendo válida en PNL(y).Se redefinen Ma(i) y Ma2(i) en forma análoga a como se hizo en la prueba del Lema-1 pero con las restricciones

$$0 \leq X(i,j) \leq 1$$

La prueba es análoga a la del Lema-1,con los subproblemas auxiliares adecuados y con:

$PNL(y)$ *en lugar de* $P(y)$*;*$MNL(y)$ *en lugar de* $M(y)$

$k=k(i)$ *en lugar de* $\forall k \in K$*.*

Lema-4: $v(PASIG) \leq v(PNL(y)) \leq v(PN(y) \leq v(P(y)) \leq v(P)$

Prueba del Lema-4:

(1) P(y) es una Relajación Lagrangiana de P con lo cual la última desigualdad a la derecha es válida.

(2) El Lema-3 y la siguiente cadena de inclusiones determinan la validez del resto de las desigualdades:

$$SF(PNL(y)) \supseteq SF(PN(y)) \supseteq SF(P(y))$$

BIBLIOGRAFIA

[1] Fisher,M.,Jaikumar,R. y Van Wassenhove,L.:A multiplier adjustment method for the assingnment problem.Management Science 32 (1986) 1095-1103.

[2] Gavish,B. y Pirkul,H.:Zero-one integer programs with few constraints-efficient branch and bound algorithms.European Journal of Operational Research 22 (1984) 35-43.

[3] Gavish,B. y Pirkul,H.:Zero-one integer programs with few constraints-efficient-lower bounding theory.European Journal of Operational Research 21 (1985) 213-224.

[4] Gavish,B. y Pirkul,H.:Efficient algorithms for solving multiconstraint zero-one knapsacks to optimality.Mathematical Programming 31,78-105 (1985).

[5] Geoffrion,A.M.:Lagrangean relaxation for integer programming.Mathematical Programming Study 2:82-114,1974.

[6] Geoffrion.A.M.y Marsten,R.E.:Integer programming algorithms:a framework and state of the art survey.Management Science,Vol 18(9):465-491.1972.

[7] Guignard,M. y Rosenwein,M.:An application oriented guide for designing lagrangean dual ascent algorithms.European Journal of Operational Research 43(1989) 917-205.

[8] Hadley,G.:Linear programming.Addison Wesley Reading Massachussetts.1962.Cap 1,pp 1-182.

[9] Jornsten,K. y Nasberg,M.: A new lagrangean relaxation approach to the generalized assingment problem.European Journal of Operational Research 27 (1986) 313-323.

[10] Ross,G.T. y Soland,R.M.:A branch and bound algorithm for the generalized assingment problem.Mathematical Programming,Vol 8(1975) pp 91-105.

[11] Ross,G.T. y Soland,R.M.:Modeling facility location problems as generalized assingment problems.Management Science,Vol 24(3) 1977,pp 345-347.

[12] Ross,G.T. y Zoltners,A.A.:Weighted assingment models and their application.Management Science Vol 25(7) pp 683-694 July 1979.

[13] Shih,W.(1979):A branch and bound method for the multiconstraint zero-one knapsack problem.J.Opl.Res.Vol 30 pp 369 to 378 (1979).

MSIM : Un Ambiente de Programación de Robots con Capacidad de Planificación de Tareas *

Jorge A. Villalobos S.

Grupo de Investigación DFAC
Departamento de Ingeniería de Sistemas y Computación
Universidad de los Andes
Apartado Aéreo 4976 - Bogotá, Colombia

Resumen: Un robot industrial se puede programar de tres formas, cada una correspondiente a un nivel de abstraccion distinto: de manera gestual, de manera textual explícita o de manera implícita. Un ambiente de programación de robots fuera de línea debe incluir un conjunto de herramientas que permita trabajar en cualquiera de los tres niveles, garantizando la compatibilidad entre ellos y entre los productos que se obtienen. MSIM por su arquitectura y diseño, constituye una propuesta de solución al problema de programacion de robots siguiendo este enfoque. Es un sistema modular, general, portable y con una estructura fácilmente extendible, que interactua amigablemente con el usuario. Ademas de una plataforma de programacion, MSIM ha sido utilizado para estudiar los problemas informáticos de implementar de manera eficiente algoritmos, como los de planificación de tareas, muy pesados computacionalmente. Una presentación formal del trabajo se puede encontrar en [VIL91] y en [MOR91].

Palabras Clave: Robótica, Programación de Robots, Planificación de Tareas, Inteligencia Artificial, Manipuladores Industriales

* Este trabajo ha sido parcialmente desarrollado por el Grupo de Robotica del Instituto de Cibernética (Universidad Politécnica de Cataluña) y parcialmente financiado por Colciencias en el marco del proyecto PIDAI (Universidad de los Andes)

1. Introducción.

Segun la definicion adoptada por la RIA (Robot Industry Association), un robot industrial es "un manipulador reprogramable multifuncional diseñado para mover materiales, piezas, herramientas, o artefactos especiales mediante movimientos variables programados para la ejecución de tareas, potencialmente muy diversas". En la industria se utiliza sobre todo para la manipulación de materiales, carga de maquinas, pintura y sellado, soldadura, mecanizado, ensamblado e inspección [SIL89].

La propiedad mas importante de los robots dentro de la automatizacion industrial es su capacidad de ser reprogramados. Esto los hace flexibles y poderosos, porque se adaptan a las necesidades de producción sin que sea indispensable modificar fisicamente las maquinas que elaboran el producto, como si ocurriria con la automatización rigida. Esta aparente ventaja se puede convertir en una desventaja si la labor de programación se vuelve demasiado dispendiosa o si los costos de reprogramar el robot resultan muy altos. Si es necesario detener una linea completa de producción durante varias semanas para hacer y probar un programa de robot, su utilización pierde rentabilidad y deja de ser una solucion atractiva.

Los ambientes de programacion de robots son un intento por resolver este problema [BEL90] [SIM89]. Para esto reenfocan toda la ingenieria tradicional de software pensando en los problemas especificos de la programacion de robots y creando herramientas de apoyo. **MSIM** fue concebido inicialmente en el Instituto de Cibernética de Barcelona [THO89] como una herramienta de programacion fuera de linea. Actualmente ha evolucionado hasta convertirse en un ambiente de programacion de robots, con un nuevo paradigma que permite al usuario manejar borradores de solución y moverse sobre diferentes niveles de abstracción en el momento de resolver cada una de las subtareas planteadas en la solucion.

El ambiente esta apoyado por un conjunto de planificadores de tareas que facilitan la labor de programacion de acuerdo al esquema planteado anteriormente. Uno de los objetivos basicos del diseño fue conseguir una estructura general, para que el ambiente pudiera trabajar con un numero considerable de robots industriales sin grandes cambios en la programación.

El objetivo del articulo es presentar de manera global el ambiente **MSIM** y se encuentra estructurado de la siguiente manera: primero se presenta la

problematica de la programacion de robots. Luego se muestra la funcionalidad y la arquitectura del ambiente propuesto como una respuesta a los problemas mostrados en la primera parte. Por último, se profundiza un poco en la manera de utilizar planificadores de tareas y lenguajes de alto nivel de abstraccion como herramientas de apoyo a la labor de programación de robots. Para una presentacion mas formal y detallada del trabajo se recomienda consultar [VIL91] y [MOR91], donde se incluye una descripción de los principales algoritmos implementados en el sistema.

2. Programación de robots.

Desde la perspectiva del robot, un programa es una secuencia de posiciones y acciones de bajo nivel (abrir y cerrar la pinza, por ejemplo) que se le especifica de alguna manera, para que el robot resuelva un problema. El robot, utilizando diferentes algoritmos de control, envía a los motores de las articulaciones las señales adecuadas para alcanzar las posiciones pedidas y alterar de ese modo la escena o realizar algun otro tipo de accion como soldar o pintar. Desde la perspectiva del programador, esta vision puede cambiar un poco. Existen diferentes niveles de abstracción en los cuales él puede expresar la tarea que debe ejecutar el robot: desde la especificacion fisica de cada posicion, hasta la formulación de la tarea a través de dos situaciones una inicial y otra final, sin indicar de manera explicita la forma de pasar de la una a la otra. Estas dos formas nos definen los límites del espectro de posibilidades que se utilizan para programar robots: programacion gestual y programacion implicita. Además de estas, existe una manera intermedia conocida como programacion textual explicita, equivalente de cierta forma a la programación imperativa en informatica.

En la programacion gestual el programador guía físicamente al robot para enseñarle la manera de ejecutar la tarea; el robot memoriza las posiciones y movimientos y luego los repite. Es la manera más facil y economica de programar un robot, pero tiene muchas limitaciones.

En la programacion textual explicita, el programador utiliza un lenguaje como VAL-II [SHI84] o LM [ITM84] con instrucciones de movimiento y, valiendose de un compilador, genera las posiciones físicas que envía al robot. Es más general y poderosa que la programacion gestual, pero todavia resulta insuficiente, como se muestra mas adelante. Existen muchos lenguajes comerciales de programación explicita de robots y su utilizacion se encuentra bastante difundida. El siguiente es un fragmento de programa escrito en VAL-

II, que toma una pieza y la lleva a otra posición, y puede dar una idea del tipo de instrucciones de estos lenguajes:

```
MOVE #PPOINT(0,90,90,60,45,30)
APPRO #POS1, -100
OPENI 50
DEPARTS -100
CLOSEI 20
MOVE #POS2
```

La programación implícita, por su parte, es el equivalente a la programación automatica en informática y todavía se encuentra en fase de investigacion. El programador especifica, con algún formalismo, la tarea que desea resolver (con dos escenas, por ejemplo), y el sistema, con algunas pautas generales y la descripcion cinemática y geométrica del robot, genera las instrucciones de movimiento que resuelven la tarea. Esta última forma de programar es muy poderosa, porque es independiente del robot, facil de mantener, rapida, etc., pero tiene varios problemas que no se han logrado resolver todavia de manera satisfactoria, sobre todo en lo concerniente a eficiencia. Existen versiones comerciales de implementaciones parciales de estos lenguajes (AUTOPASS [LIE77], LAMA [LOZ77]) pero, debido a las grandes restricciones que tienen, son muy poco utilizados comercialmente. El siguiente es un fragmento de codigo utilizado por AUTOPASS para ensamblar dos objetos y colocarlos sobre otro: especifica tres tareas, pero no se refiere en ningún momento a las posiciones que debe alcanzar el robot para resolverlas:

```
GRASP OBJETO1
INSERT OBJETO1 INTO OBJETO2
PLACE OBJETO2 ON OBJETO3
```

Independiente de la forma que se haya escogido para programar el robot existe un problema que no se ha considerado y que genera grandes complicaciones: la incertidumbre acerca del estado del mundo. En los ambientes industriales, que son bastante controlados y poco variables, esta incertidumbre disminuye, pero aun asi, es uno de los factores mas dificiles de manejar en el proceso de programación. ¿Qué pasa si la pieza que debe agarrar el robot se encuentra desplazada unos milimetros de donde deberia estar?. El robot podria chocar contra ella y producir resultados inesperados y peligrosos para toda la linea de producción. Es indispensable, entonces, que el robot este recibiendo constantemente información sobre el estado del mundo y la manera como va evolucionando la tarea. Para esto se han integrado a los robots

sensores de vision, de fuerza, de proximidad, de tacto, etc. que lo ayudan a identificar situaciones potencialmente peligrosas y poder, asi, adoptar a tiempo las acciones correctivas necesarias. También puede utilizar la informacion sensorial para corregir en ejecucion algunas posiciones fisicas del programa que no corresponden con la realidad.

El problema es, ¿cómo integrar esta nueva información en el momento de hacer el programa?. En programacion gestual no hay manera de hacerlo y, por eso, se considera tan limitada. En la programación textual explícita se debe escribir un programa aparte para el manejo de los sensores que se ejecute en paralelo con el programa del robot y que de forma sincronizada informe al otro lo que va percibiendo del mundo. Esto no solo no es sencillo, sino que en muchos casos es más complicado que la misma programación del robot, exigiendo al programador bastante conocimiento en estrategias de movimiento basadas en información sensorial y reduciendo por tanto el personal capacitado para llevar a cabo esta labor. En programacion implícita existen varios enfoques distintos sobre la manera de utilizar la información sensorial, pero todos incluyen la generación automática del programa de sensores y su sincronización con el programa del robot. Algunos sistemas, como el que se presenta, van más allá y son capaces de replanificar partes de la tarea según las condiciones especiales que se vayan detectando mediante los sensores.

La programacion de un robot se puede hacer en linea, utilizando directamente el robot, o fuera de linea, utilizando herramientas CAD, simuladores graficos y validadores de programas. Por costo, comodidad y seguridad es mas conveniente esta ultima opcion. MSIM está diseñado para la programacion fuera de linea de robots, utilizando los tres niveles de abstraccion y permitiendo que se combinen los resultados obtenidos desde cualquiera de ellos.

3. Funcionalidad del ambiente.

Desde un punto de vista puramente funcional, el ambiente está compuesto por cuatro módulos que se comunican y sincronizan a traves de un coordinador, como se muestra en la figura 1.

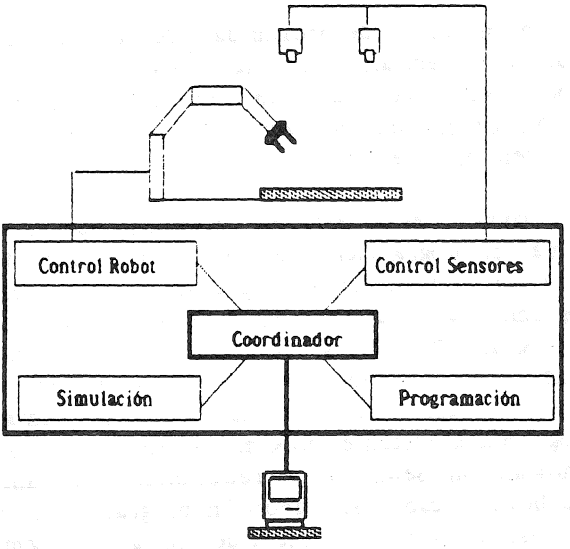


figura 1.- Estructura funcional de MSIM

El coordinador, además de interactuar con el usuario, mantiene un modelo actualizado del robot y de su entorno, y administra el flujo de información y de control entre los diferentes modulos. El modulo de comunicación con el robot es el responsable de enviar los comandos adecuados al controlador físico del robot para que ejecute un movimiento. Este módulo es dependiente del robot que se utiliza y se debe escribir para cada robot que se quiera manejar desde el ambiente. El módulo de sensores, por su parte, recibe peticiones del coordinador para establecer algún hecho del entorno y retorna la informacion pedida. Por ahora funciona con sensores monovaluados (retornan un solo valor, como puede ser la proximidad a un objeto), pero se busca tambien integrar sensores como los de visión sobre los cuales es necesario hacer reconocimiento de formas antes de responder el requerimiento del coordinador. El problema de este último tipo de sensores es el tiempo que gasta un algoritmo en reconocer, a partir de una imagen, la posición y orientación de un objeto de la escena, o alguna otra característica de dicho cuerpo. Es importante recalcar que toda la sincronización entre el robot y los sensores pasa por el coordinador.

El módulo de programación tiene la estructura interna mostrada en la figura 2. que corresponde a los tres niveles de programacion presentados anteriormente.

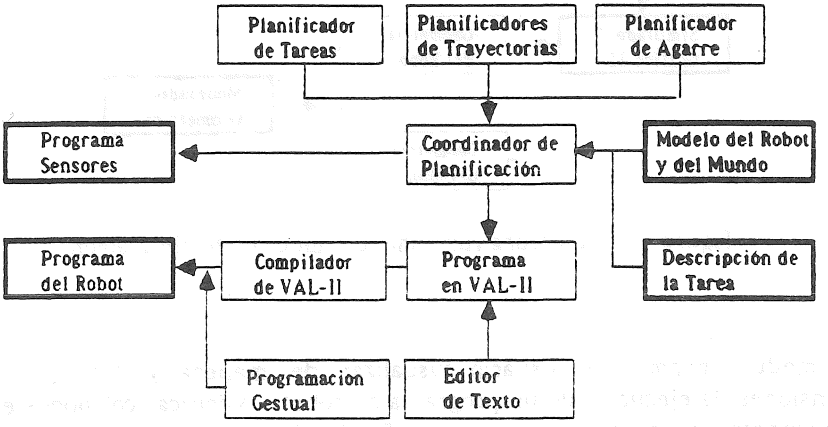


figura 2.- Estructura funcional del módulo de programación

El programador puede trabajar de manera gestual apoyandose en el simulador, de manera textual explícita utilizando el editor de texto del ambiente o de manera implícita generando un plan global de solución y resolviendo cada subtarea utilizando los planificadores diponibles. En [MOR91] aparece una propuesta de lenguaje de planes basado en rasgos y un esquema de interacción y traducción hacia VAL-II, incluyendo la manera de generar el programa de sensores. Se presenta también la forma de replanificar subtareas, cuando la información sensorial indica la inaplicabilidad de la solución encontrada anteriormente por un planificador. Por ejemplo, si la pieza que se desea agarrar se encuentra rotada algunos grados y por esta rotación la pinza no puede alcanzar el punto de agarre que se había seleccionado (porque colisiona contra otra pieza del entorno, por ejemplo), el coordinador pide a los planificadores otras soluciones aplicables a las nuevas condiciones y continúa con la ejecucion. Mediante la utilizacion del lenguaje de planes propuesto, el programador puede manejar borradores de solución y trabajar de lo abstracto a lo concreto facilitando de esta manera la labor de programación.

Por otra parte, existe el módulo de simulacion y validación de programas que tiene la estructura funcional mostrada en la figura 3.

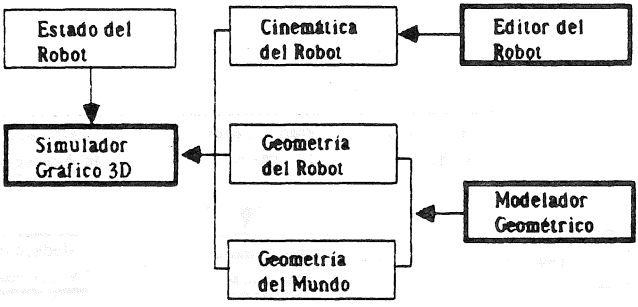


figura 3.- Estructura funcional del módulo de simulación

Este módulo permite al usuario visualizar de manera gráfica 3D (tres dimensiones) la ejecución de un programa de robot y verificar colisiones entre los elementos presentes en la escena. El simulador cuenta con el modelo cinematico del robot definido mediante los parámetros de Denavit-Hartenberg [SIL89] y con los modelos geométricos del robot y de su entorno aproximados por poliedros de caras planas definidos con CGS (geometría constructiva de solidos) y con una representación por la frontera [AKM88]. Toma como entrada codigo generado por el compilador de VAL-II y hace una simulación gráfica del movimiento del robot. Mantiene desplegada información del estado del robot, permite situar de manera interactiva al observador, tener visualizacion en alambre o realista, colocar al observador en la pinza para visualizacion de aproximacion y agarre, etc. Cuenta, además, con un sistema de ayuda en linea.

4. Arquitectura del ambiente.

Desde un punto de vista informático, MSIM fue construido con una arquitectura de capas sobre un modelo orientado por objetos [MEY88]. Esto facilita incluir en él nuevas opciones y facilidades, sin necesidad de modificar lo ya existente. También permite manejar múltiples vistas del mismo modelo, según las necesidades, aumentando así las posibilidades de expresar problemas de diferentes tipos, en diferentes términos.

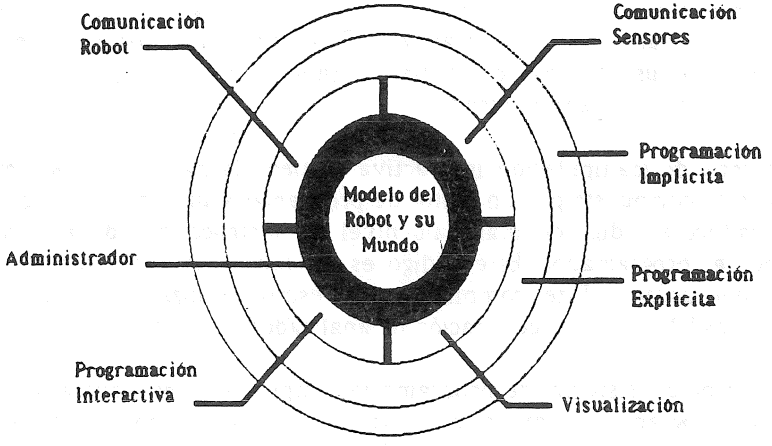


figura 4.- Arquitectura informática de MSIM

El nucleo del sistema es un modelo operacional del robot y de su mundo, que simula su comportamiento. Eso quiere decir que responde exactamente igual a como respondería en la realidad el robot a estímulos externos (instrucciones del programador, choque con una pieza, etc.). Este modelo maneja un primer nivel de razonamiento geométrico [PER86] que permite al robot agarrar piezas, transportar ensambles, decidir si hay una colision, etc., y constituye el soporte de todo el ambiente.

Cubriendo este núcleo aparece una primera capa, que corresponde al administrador del modelo, encargado de la interacción del núcleo del sistema con el robot físico, los sensores y el usuario, y cuya misión es garantizar, en todo momento, la coherencia entre la situación actual del robot, lo que perciben los sensores del entorno del robot y lo que ve y quiere hacer el usuario.

La comunicación con el usuario se encuentra en la segunda capa del sistema, en relación directa con el administrador del núcleo; se puede dividir en dos partes: visualización y programación gestual o interactiva. La visualización mantiene informado al programador sobre el estado del modelo utilizando una representación gráfica 3D. De esta forma el usuario obtiene una visión dinámica, completa y actualizada de la escena, incluyendo la información obtenida por los sensores. La manipulación gestual permite al usuario operar

directamente sobre el robot, enviarle órdenes de bajo nivel, mover sus articulaciones, manipular las piezas, etc. Esta capa toma todos los eventos generados por el usuario (mover un dial, oprimir un botón, etc.), los interpreta y comunica la orden al administrador.

Sobre la capa de manipulación interactiva se tiene la parte de programación explícita; está compuesta por un editor de programas y un compilador de VAL-II que traduce a código de más bajo nivel las instrucciones de movimiento dadas por el programador. Este código es enviado a la capa inferior como ordenes del usuario. Este compilador fue escrito utilizando YACC y LEX, herramientas UNIX para la generación de analizadores sintácticos y léxicos.

La última capa del sistema, de programacion implícita, se encuentra localizada sobre la de programación textual explícita. Está compuesta por un conjunto de planificadores que generan código VAL-II y se explican en detalle en la siguiente seccion.

MSIM está escrito en C utilizando la librería gráfica PHIGS. Es completamente portable entre máquinas UNIX y actualmente se encuentra funcionando en estaciones gráficas Apollo, Sun e IBM-RT. Existe también una versión en C++ 2.0 y una versión sobre la librería gráfica CORE de Sun, utilizadas para estudiar los problemas de mantenimiento y portabilidad de ambientes gráficos orientados por objetos.

5. La Planificación como Herramienta de Programación.

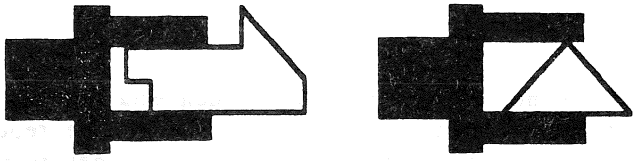
Para escribir un programa en MSIM, el usuario sigue el siguiente proceso: crea en el editor de planes una primera aproximación de la solución -llamada un borrador o un bosquejo- en la cual, a grandes rasgos, resuelve el problema partiéndolo en subtareas más sencillas. Luego, comienza a resolver cada subtarea y a ajustar el plan global a medida que se vayan concretando las soluciones parciales, utilizando todas las herramientas del ambiente. En particular puede utilizar los planificadores que son capaces de sugerir de manera interactiva soluciones a tareas sencillas. Toda la coherencia entre el borrador y sus partes la garantiza el sistema, reescribiendo el borrador hasta llegar a la solución expresada en un lenguaje que se puede considerar de nivel implícito.

El ambiente cuenta con cuatro planificadores: dos para encontrar trayectorias libres de colisión (uno para movimientos grandes y otro para movimientos finos), un planificador de agarre y un planificador de tareas simples.

El planificador de movimientos finos sin colisión está basado en la idea de potenciales. En este algoritmo se simula una fuerza de repulsión de los obstáculos y una fuerza de atracción hacia la posición final, de manera que el robot navega por ese campo de potenciales evitando las piezas de la escena y aproximándose hacia la posicion final. El algoritmo esta apoyado por heurísticas que impiden que el robot se quede en mínimos locales. La funcion de potencial fue una modificacion de la propuesta en [HWA88]. Este planificador ha demostrado su eficiencia para hacer movimientos cortos pues genera trayectorias suaves y seguras, de manera rápida y a un costo computacional bajo.

Para los movimientos grandes, se cuenta con un planificador de trayectorias que trabaja sobre un espacio de configuraciones discretizado y aproximado [LOZ83] [LOZ87]. La idea de fondo es cambiar el espacio de búsqueda de soluciones a un espacio generado por los tres primeros grados del libertad del robot (los de posicionamiento) aproximando los demas grados de libertad (los de orientacion) a traves de una esfera que los incluya. De esta manera el problema se reduce a encontrar la manera de mover un punto en un espacio de 3 dimensiones donde cada eje corresponde a los valores de una articulación. Este planificador requiere grandes recursos computacionales y toma algun tiempo en encontrar una solución debido, más que todo, a que los algoritmos para el cambio de espacio tienen altas complejidades. El planificador muestra graficamente la solucion encontrada y le presenta al programador toda la informacion asociada con la trayectoria. para que este pida la generacion de codigo en VAL-II, solicite otra solucion, o fije restricciones adicionales.

El planificador de agarre es capaz de sugerir al programador diferentes puntos para tomar una pieza, ya sea para ensamblarla o para transportarla [THE88] [ALA89]. Trabaja para pinzas con dedos paralelos y funciona buscando posiciones estables. Localiza todas las parejas de caras paralelas cuyas proyecciones de una sobre la otra se intersecten. Busca tambien parejas cara-arista que sean estables. Este planificador está implementado con un algoritmo de complejidad lineal y utiliza como estructura de representacion interna una esfera de Gauss tal como se sugiere en [GAT89]. El planificador esta completamente parametrizado, de manera que el usuario determina el nivel de precisión al cual debe trabajar, el tipo de soluciones que le interesa, el grado mínimo de estabilidad, etc.



Su funcionamiento es totalmente interactivo, en el sentido que después de establecer las condiciones exigidas por el usuario comienza a mostrarle gráficamente cada uno de los posibles puntos de agarre con su respectiva dirección de aproximación libre de colisión. Cuando el usuario escoge una solucion, el planificador genera el código VAL-II necesario para hacer ese agarre.

El planificador de tareas simples resuelve problemas de movimiento de piezas en la escena del robot [ALA89]. Para esto divide toda tarea en tres subtareas más sencillas: agarrar la pieza, transportarla y dejarla en el sitio pedido. Luego utiliza los otros planificadores para encontrar soluciones parciales y las combina hasta encontrar una solución viable a la tarea global. Tan pronto encuentra una solución se la presenta gráficamente al programador y, si éste lo desea, genera el código correspondiente en VAL-II. Su funcionamiento tambien es parametrizable, permitiendo al programador especificar el tipo de solucion que busca o adicionandole restricciones. Puede, por ejemplo, pedir que el cuerpo sea agarrado únicamente por caras paralelas con un factor de estabilidad superior a algun valor, o que, para transportarlo, se busque la ruta mas corta o la mas segura, etc.

6. El Futuro de MSIM.

En este momento se trabaja en varios frentes sobre **MSIM**. Como primera medida se están incorporando al ambiente dos nuevos robots diseñados en la Universidad y actualmente en proceso de fabricación (Andes-1 y Andes-2). Se está terminando la implementación del lenguaje implícito basado en rasgos descrito en [MOR91] y mencionado en este artículo y se están diseñando nuevas facilidades en lo que respecta al manejo de borradores durante la fase de programación. Se trabaja en la implementación de algoritmos de reconocimiento de formas para incorporación de sensores no monovaluados. Se estudian nuevas estrategias de replanificación en línea [BAS89], que sean más eficientes y más generales, para que funcionen en un mayor numero de casos. Se esta implementando otro nuevo planificador de trayectorias, que sea mas preciso y que permita trabajar en ambientes altamente congestionados.

7. Conclusiones.

En este articulo se presenta **MSIM**, un ambiente de programación de robots que permite trabajar al usuario en tres diferentes niveles de abstracción, cada uno correspondiente a una forma de programar un robot: programación gestual, programacion textual explicita y programación implicita. Incluye además un nuevo paradigma para la programación de robots que permite al usuario manejar borradores de soluciones, apoyarse en planificadores para resolver algunas subtareas, y moverse sobre los tres niveles de programacion para plantear la solución completa de un problema. Se muestra la estructura funcional e informática de **MSIM** como una propuesta general para este tipo de ambientes.

Además de la construcción de **MSIM**, el trabajo ha servido para estudiar y plantear algunas soluciones a los problemas informáticos típicos en ambientes gráficos para la programación fuera de línea de máquinas industriales. Tambien ha permitido estudiar la forma de aplicar la metodología de programación orientada por objetos a ambientes gráficos dinámicos, en donde las necesidades de eficiencia son un factor determinante en la estructura del software.

8. Agradecimientos.

El trabajo que se presenta en este artículo es el producto del esfuerzo de muchas personas entre las cuales se deben destacar el profesor Federico Thomas del Instituto de Cibernética y los ingenieros Eduardo Villegas, Jorge Morales y Oscar Ramos de la Universidad de los Andes.

9. Bibliografía.

[AKM88] Akman, V., Geometry and Graphics Applied to Robotics, Theoretical Foundations of Computer Graphics and CAD, Springer-Verlag, 1988.

[ALA89] Alami, R., Simeon, T., Laumond, J., A Geometrical Approach to Planning Manipulation Tasks. The Case of Discrete Placements and Grasps, septiembre 1989.

[BAS89] Basañez, L., Operation Specialists for Automatic Programming and Monitoring of Robotic Assembly Cells, Journal of Robotics and Computer-Integrated Manufacturing, 1989.

[BEL90] Bellier, C., VCP: Un Module Interactif de Verification et Correction de Programme en Robotique d'Assemblage, Rapport de Recherche, LIFIA, Institut IMAG, febrero 1990.

[GAT89] Gatrell, L., CAD-Based Grasp Synthesis Utilizing Polygons, Edges, and Vertexes, IEEE, 1989.

[HWA88] Hwang, Y., Path Planning Using Potential Field Representation, IEEE, 1988.

[ITM84] ITMI (Industrial Technology and Machine Intelligence), LM Reference Manual, 1984.

[LIE77] LLieberman, L., AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly, IBM Journal of Research and Development, Vol. 21, No. 4, julio 1977.

[LOZ77] Lozano-Perez, T., LAMA: A Language for Automatic Mechanical Assembly, M.I.T,1977.

[LOZ83] Lozano-Perez, T., Spatial Planning: A Configuration Space Approach, IEEE Trans. on Computers, Vol. C-32, No. 2, febrero 1983.

[LOZ87] Lozano-Perez, T., A Simple Motion-Planning Algorithm for General Robot Manipulators, IEEE Journal of Robotics and Automation, Vol. RA-3, No. 3, junio 1987.

[MEY88] Meyer, B., Object Oriented Software Construction, Prentice-Hall, 1988.

[MOR91] Morales, J., Un Lenguaje Implicito Basado en Rasgos para la Programacion de Robots, Universidad de los Andes, 1991.

[PER86] Pertin, J., Modelisation du Raisonnement Geometrique pour la Programmation des Robots, Tesis de Doctorado, Institut National Polytechnique de Grenoble, 1986.

[SHI84] Shimano, B., VAL-II: A New Robot Control System for Automatic Manufacturing, Proc. of the Int. Conference on Robotics, marzo 1984.

[SIL89] Silva, M., Introduccion a la Robotica Industrial, Universidad de Zaragoza, 1989.

[SIM89] Simeon, T., Generation Automatique de Trajectoires sans Collision et Planification de Taches de Manipulation en Robotique, Tesis de doctorado, Université Paul Sabatier de Toulouse, enero 1989.

[THE88] Theveneau, P., Utilisation du Raisonnement Geometrique pour la Planification en Robotique d Assemblage: Le Systeme Pamela, Tesis de Doctorado, Institut National Polytechnique de Grenoble, 1988.

[THO89] Thomas, F., Paquete Gráfico para la Programacion Fuera-de-linea y Simulación de un Robot PUMA 560, Instituto de Cibernética, 1989.

[VIL91] Villegas, E., Algoritmos Eficientes para Resolver los Problemas de Visualización, Colision y Planificacion en un Ambiente de Programación de Robots, Universidad de los Andes, 1991.

CARACTERISTICAS DETERMINANTES DAS TECNOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE

Vera Maria Benjamim Werneck
Teresa Cristina de Aguiar
Ana Regina C. da Rocha

COPPE/UFRJ (SISTEMAS)

Caixa Postal 68511
CEP 21945 - Rio de Janeiro - RJ

Tel: (5521) 590-2552
Fax: (5521) 290-6626
E-Mail: cos99300@UFRJ.BITNET

RESUMO

Os problemas que vêm sendo enfrentados por gerentes e equipes de desenvolvimento de software e também a necessidade de se obter produtos de melhor qualidade, possibilitam e justificam a proposta de existência de um ambiente de desenvolvimento de software (ADS).

É importante no entanto que, ao se projetar um ADS, sejam consideradas as particularidades do software que será desenvolvido. Projetar ADSs com essa filosofia é o objetivo do Projeto TABA [Rocha 89 e Rocha 90], que prevê a configuração de ADSs para atender às características das diferentes áreas de aplicação ou dos requisitos próprios de cada projeto.

Este artigo descreve um sub-projeto do TABA, realizado com o objetivo de estabelecer um relacionamento entre as características de projetos de software e algumas tecnologias utilizadas na sua implementação.

Foi realizada uma classificação de domínios de aplicações, com a finalidade de adquirir conhecimentos sobre as aplicações. A partir dessa classificação realizou-se uma pesquisa de campo junto

a profissionais de desenvolvimento de software, estabelecendo o relacionamento entre as características das aplicações e as tecnologias por elas determinadas. Esse artigo apresenta o estudo realizado sobre aplicações, a pesquisa de campo e os resultados obtidos nessa pesquisa.

1. INTRODUÇÃO

A introdução do uso dos computadores nas diversas áreas de atividades aumentou a complexidade e o tamanho dos sistemas desenvolvidos e gerou novos problemas que vêm sendo enfrentados por gerentes e equipes de desenvolvimento de software.

Não apenas os problemas, mas também a necessidade de se obter produtos de melhor qualidade, possibilitam e justificam a proposta de existência de um ambiente de desenvolvimento de software (ADS). Os métodos usados permitem sistematizar as atividades de desenvolvimento de software, enquanto sua automação incrementa a produtividade, o gerenciamento, a facilidade de manutenção e a possibilidade de reutilização [Wasserman 82].

É importante no entanto que, ao se projetar um ADS, sejam consideradas as particularidades do software que será desenvolvido. Projetar ADS com essa filosofia é o objetivo do Projeto TABA [Rocha 89 e Rocha 90], que prevê a configuração de ADSs para atender às características das diferentes áreas de aplicação ou dos requisitos próprios de cada projeto.

Este artigo descreve um sub-projeto do TABA inserido neste contexto, realizado com o objetivo de estabelecer um

relacionamento entre as características de projetos de software e algumas tecnologias utilizadas na sua implementação. Os resultados deste trabalho forneceram subsídios para a construção do especificador de ambientes de desenvolvimento de software da Estação TABA.

O objetivo do especificador de ADSs é apoiar o engenheiro de software na definição de ambientes adequados a projetos ou a aplicações específicas, de acordo com suas particularidades e experiências anteriores [Rocha 89a]. Esta, entretanto, não é uma tarefa trivial. A especificação de um ambiente é um processo heurístico ou seja, critérios de escolha não são definidos explícita e claramente para todos os casos, necessitando do conhecimento de especialistas [Blum 86]. Estes especialistas devem ter conhecimento da área de aplicação e experiência no uso de métodos e ferramentas, o que muitas vezes não pode ser encontrado em um único indivíduo.

Na Estação TABA, a especificação do ambiente de desenvolvimento poderá ser feita diretamente pelo usuário, através de uma linguagem para definição de ambientes ou através de um sistema especialista de suporte à decisão (DIAG) que auxilia na tomada de decisões sobre os componentes desejáveis para um ADS tendo em conta as características do produto a ser desenvolvido [Rocha 89a].

De forma a construir a base de conhecimentos do DIAG foi realizada uma classificação de domínios de aplicações, com a finalidade de adquirir conhecimentos sobre as aplicações. A partir dessa classificação realizou-se uma pesquisa de campo junto a profissionais de desenvolvimento de software, estabelecendo o

relacionamento entre as características das aplicações e as tecnologias por elas determinadas. Esse artigo descreve o estudo realizado sobre aplicações e os resultados obtidos na pesquisa de campo.

2. APLICAÇÕES: COMPONENTES E TECNOLOGIAS

Nesse estudo pretendeu-se conhecer melhor o que seja uma aplicação de computador e sua relação com sua implementação computacional. Para se conhecer e dominar um corpo de conhecimento é necessário desmembrá-lo e conhecer suas partes e suas engrenagens.

A aplicação computacional é uma adaptação de funções ou tarefas, que serão executadas em parte ou totalmente pelo computador. A aplicação desempenha no computador funções, atendendo a uma ou mais áreas de um determinado contexto do mundo, com a vantagem de se ter grande capacidade de memória e do processamento ser mais rápido que a capacidade humana.

Adaptar não significa copiar. É entender o problema, organizá-lo, verificar soluções possíveis e, por fim, implementá-lo para ser compreendido pelo computador e seu usuário.

Pode-se definir a aplicação computacional como a modelagem de uma realidade, com propriedades definidas segundo abstrações, utilizando recursos e técnicas computacionais. O resultado dessas abstrações é um modelo que simula no computador a realidade estudada.

A atuação do computador nas diversas áreas da sociedade tem sido gradativa e constantemente expandida. Muitas vezes a

tecnologia disponível no momento não permite a implementação total ou mesmo parcial de uma tarefa. O desenvolvimento de uma nova tecnologia aumenta a possibilidade de informatização de novas áreas da sociedade e aprimora aplicações já disponíveis.

Assim a aplicação está sujeita a mudanças pela evolução da tecnologia e dos conceitos da própria área. Normalmente a função básica a que se destina o software não muda com a introdução de novas técnicas computacionais, sofrendo somente um melhor ajustamento à realidade através de maiores facilidades, como por exemplo: melhor desempenho, maior alcance das informações, maior disponibilidade.

2.1 Componentes das Aplicações

Foram estudados alguns modelos de custos e de controle de qualidade, com a finalidade de se indentificar os componentes das aplicações, encontrando-se características gerais, pois para aferir sobre o custo e a qualidade é necessário se conhecer o produto avaliado.

A partir dos atributos e das classificações de aplicações encontrados na literatura técnica foram identificados e agrupados os seguintes componentes: características básicas, requisitos de qualidade e características de tamanho.

Características Básicas

As características básicas são o conjunto de atributos funcionais, de identificação do produto e sua organização, pois o produto está diretamente relacionado com o seu ambiente organizacional.

Nessa classe foram criados dois grupos distintos, o primeiro relaciona os atributos funcionais e é composto pela complexidade do trabalho, os tipos de transações, quantidade e complexidade dos cálculos, os tipos de entrada e de saída, tempo de resposta e a relação da aplicação com o tempo de resposta. A outra classe tem atributos relacionados com a aplicação, o ambiente operacional e a organização onde se encontra a aplicação. O tipo, tamanho e grau de desenvolvimento da empresa influenciam, diretamente, no sistema computacional. Nesse grupo estão os fatores de custo, de grau de interação com o usuário, de grau de integração com outras aplicações, de influência do custo e do prazo no desenvolvimento do software.

Características de Qualidade

A qualidade de um software deve ser medida comparando-o com seus requisitos de qualidade, isto é, com os fatores considerados importantes para o produto e que são definidos durante a fase de especificação de requisitos. Esses atributos variam em função da aplicação. Diferentes aplicações determinam diferenças nos requisitos necessários e no grau em que devem ser atingidos. Esses requisitos são, normalmente, predominantes na escolha de um ambiente de desenvolvimento de software.

Foram selecionados na pesquisa de campo, os requisitos de qualidade mais significativos para a maioria das aplicações: Confiança, Desempenho, Facilidade no Uso do Sistema, Tempo de Aprendizagem de uma Aplicação, Disponibilidade, Satisfação do Usuário Final, Precisão, Nível de Segurança, Administração de Erros, Dependência do Hardware e Dependência do Software Operacional.

Características de Tamanho

As características de tamanho são os fatores que estão relacionados diretamente com o tamanho do software executável no computador, são fatores que identificam a capacidade do sistema computacional implementado. Seu tamanho (porte do sistema), seu volume de dados de entrada e de saída e volume de dados permanentes.

2.2 Tecnologias

Conforme mencionado anteriormente, o processo de desenvolvimento de um software é significativamente dependente das tecnologias adotadas em sua implementação.

A própria escolha dos métodos/ferramentas parte principalmente da técnica de processamento utilizada. O uso de técnicas específicas, como a de Banco de Dados, altera também o ambiente.

A princípio, na identificação do problema, um engenheiro de software especializado tem uma idéia do tipo de processamento e do uso de técnicas específicas. Isso se dá considerando vários fatores, desde a natureza do problema, ambiente operacional do Centro de Processamento de Dados, como também aspectos da organização.

Em aplicações numéricas o uso de modelos matemáticos específicos também influencia as características gerais da aplicação e o seu processo de desenvolvimento.

As tecnologias consideradas nesse trabalho são as seguintes: banco de dados, tipo de processamento, teleprocessamento e métodos numéricos.

3. PESQUISA DE CAMPO

A partir do estudo da literatura foi feita uma adaptação da metodologia de pesquisa utilizada por Werneck [Werneck 89], e elaborado um questionário para a pesquisa de campo.

Numa primeira identificação das características das aplicações, foi formulado o objetivo da pesquisa de campo que pretendeu verificar as seguintes hipóteses: existem características básicas inerentes às aplicações, e existem diversos fatores relacionados à implementação computacional da aplicação, sendo que esta é diferente dependendo do ambiente e da tecnologia empregada;

Foram identificados alguns atributos essenciais das aplicações e suas implementações sendo elaborado um questionário composto de duas partes. A primeira parte foi aplicada a gerentes de desenvolvimento de software para se obter a estrutura de informática das empresas e suas aplicações mais importantes. Na segunda parte, teve-se contato direto com os reponsáveis pelo desenvolvimento ou manutenção das aplicações e as perguntas referem-se às características funcionais, de implementação e de requisitos de qualidade.

O universo da pesquisa de campo foi limitado a três áreas: administrativa, científica (ciências exatas), e industrial. Essas áreas foram escolhidas considerando-se aspectos de maturidade no

uso do computador, da tecnologia empregada e de interesse ao projeto TABA.

A escolha das empresas e setores a serem pesquisados considerou alguns critérios [Werneck 90], sendo selecionados os seguintes setores: (a) distribuição de petróleo, (b) serviços de transportes e (c) química e petróleo.

Após a fase a validação da metodologia de pesquisa alguns ajustes foram feitos e o questionário foi testado em duas empresas para verificar sua aplicabilidade e praticidade como ferramenta.

A coleta dos dados foi efetuada pela aplicação dos questionários nas diferentes firmas, cada um se relacionando a sistemas desenvolvidos ou mantidos por essas empresas. Foram eliminados da entrevista, os pacotes de sistemas onde a manutenção era dada por empresas de consultoria ou de serviços.

Concluída a fase de pesquisa foi feita uma análise dos dados e aplicação estatística, utilizando-se a Análise Estatística do pacote SPSS/PC [SPSS 84].

A análise estatística do qui-quadrado foi utilizada para o teste de independência entre as duas variáveis, através da função "CROSSTABS" do sistema SPSS/PC. Esta função elabora a tabela de contigência dos dois atributos e fornece o coeficiente de significância que possibilita rejeitar a hipótese caso esse seja suficientemente pequeno (menor que 0,00005). O valor do lambda simétrico e assimétrico é calculado e este tem valor igual a zero na independência das variáveis, não sendo necessariamente verdadeira a recíproca.

Após essa etapa foi realizada a análise estatística descritiva de cada atributo correlacionado com a tecnologia, utilizando-se a função "FREQUENCIES" do sistema SPSS/PC, considerando grupos distintos a cada tipo de tecnologia.

4. RESULTADOS: ATRIBUTOS DETERMINANTES DAS TECNOLOGIAS

A Tabela 1 fornece a distribuição da amostra em termos do número de sistemas e de empresas entrevistadas por setor.

TABELA 1 - DISTRIBUIÇÃO DE SISTEMAS POR EMPRESAS E SETOR

SETOR	Código	Número de Empresas	Sistemas	
			Total	Entrevistados
Serviços de Transporte	ST	6	220 100%	68 30,9%
Distribuição de Petróleo	DP	5	200 100%	59 29,5%
Química e Petróleo	QP	5	339 100%	70 20,7%
TOTAL		17	759 100%	197 26%

No universo de aplicações foram encontradas de forma significativa as seguintes tecnologias: banco de dados, teleprocessamento e método numérico. Os tipos de processamento identificados foram "batch", "on-line" e tempo real.

A hipótese a ser verificada nesse trabalho foi a existência de fatores relacionados com a implementação computacional da aplicação, analisando a correlação entre a tecnologia utilizada e os atributos importantes para determinação dessas técnicas. Foi testada essa hipótese através do teste de independência do qui-quadrado, conforme descrito anteriormente.

Banco de Dados

Foram testadas a relação de dependência entre o uso de banco de dados e os atributos: tipo de setor, tipo de empresa, volume de dados de entrada e de saída, volume de dados armazenados em memória permanente, porte do sistema, grau de integração com outras aplicações e quantidade de transações de consulta, de atualização e de emissão de relatórios.

O tipo de empresa tem correlação com o uso de banco de dados, embora essa relação seja em um sentido, significando que esse uso é mais frequente num determinado tipo de empresa.

O uso de banco de dados é mutuamente dependente do setor do mercado que a empresa atua, podendo ser explicado pelo grau de competitividade do setor ou pela natureza da tecnologia que agiliza a rede de informações da empresa como também possibilita uma maior integração das diversas áreas e controle dessas informações.

Foi verificado (Tabela 2) que o uso de banco de dados tem correlação com o volume de dados de entrada e de saída, o porte do sistema, o grau de integração com outras aplicações e a quantidade de transações de tipos de emissão de relatórios, embora essa relação não seja nos dois sentidos. O volume de dados permanentes e a quantidade de transações de atualização são mutuamente dependentes do uso de banco de dados, ou seja, a relação de dependência ocorre nos dois lados.

Pode ser comprovado que não existe relação entre o uso de banco de dados e a quantidade de transações de consulta.

TABELA 2 - ATRIBUTOS DETERMINANTES DO USO DE BD

Quantidade Sistemas	Com BD 100			Sem BD 97		
SETOR MERCADO: (%)	ST 27	DP 33	QP 40	ST 42,2	SP 26,8	QP 31
TIPO DE EMPRESA: (%)	E 39	B 28	M 33	E 62,9	B 19.6	M 17,5
COMPLEXIDADE:	MA/ME			MA/B		
TRANSAÇÕES (%): .Atualização	24,7			24,1		
.Relatório	21,5			26,8		
Grau Integração com outras Aplicações:	E/B			MA/MB		
PORTE (kLS)	768			1610		
VOLUME DE DADOS ENTRADA (Mb)	426			22209		
VOLUME DE DADOS SAÍDA (Pag/Mes)	21270			159743		
VOLUME DE DADOS PERMANENTES (Mb)	368			2040		

Escala de Valores :
- 5 - E - Extremamente Alto
- 4,5 - MA - Muito Alto
- 4 - A - Alto
- 3 - ME - Médio
- 2 - B - Baixo
- 1 - MB - Muito Baixo
- 0 - NE - Nenhum ou Ausente

Tipo de Empresa:
- E - Estatal
- B - Brasileira
- M - Multinacional

Setor do Mercado:
- ST - Serviços de Transporte
- DP - Distribuição Petróleo
- QP - Química e Petróleo

Tipo de Processamento

A relação de dependência entre os tipos de processamento e os fatores foram analisados para os seguintes atributos: primitiva e área de aplicação, tipo de setor, tipo de empresa, grau de interação com o usuário, quantidade de transações de consulta, de atualização e de emissão de relatórios, relação com o tempo, tempo

de resposta do sistema, volume de dados de entrada e de saída, volume de dados armazenados em memória permanente, porte do sistema e grau de desempenho do sistema.

Não existe relação entre o tipo de empresa e o tipo de processamento.

O setor da empresa tem correlação com o tipo de processamento.

Foi verificado (Tabela 3) que o tipo de processamento dos dados tem correlação com a área de aplicação, a quantidade de tipos de transações de atualização, o volume de dados de entrada e de saída, o volume de dados permanentes e o grau de desempenho do sistema, sendo essa relação mutuamente dependente.

A primitiva de aplicação, o grau de interação com o usuário, a quantidade de transações de consulta, a relação com o tempo, o tempo de execução do sistema e o porte do sistema não têm relação de dependência com o tipo de processamento.

A quantidade de transações de emissão de relatórios depende do tipo de processamento, numa relação unilateral, isto é coerente pois essa função é realizada, normalmente, por processamento "batch" e este tipo de processamento não necessariamente emite relatórios.

TABELA 3 - ATRIBUTOS DETERMINANTES DO TIPO DE PROCESSAMENTO

	Batch	OnLine	Tempo Real (TR)	Batch + Online	OnLine + TR	Batch+ Online +TR
Quantidade Sistemas	48	26	5	96	15	7
ÁREA (%) Admnistrat	77,1	76,9	20	82,3	73,3	57,1
DA Científica	16,7	3,4	0	7,3	0	0
APLICAÇÃO Industrial	6,3	15,4	80	8,3	26,7	42,9
TRANSAÇÕES (%):						
.Atualização	26	26	18,3	23,1	25,7	27,9
.Relatório	31,9	17,4	21,4	23,1	19,6	22,3
VOLUME DE DADOS ENTRADA (Mb)	504	783	99	22188	125	9
VOLUME DE DADOS SAÍDA (PAG/MES)	23786	16103	3506	59689	684428	49538
VOLUME DE DADOS PERMANENTES(Mb)	175	1491	2	1848	24	1455
QUALIDADE.Desempenho	MA/B	E/ME	E/A	E/ME	A	MA/B
TEMPO DE EXECUÇÃO DO "ON-LINE" (Seg)	1801	23	25	108	6	16

Escala de Valores :
5 - E - Extremamente Alto
4,5 - MA - Muito Alto
4 - A - Alto
3 - ME - Médio
2 - B - Baixo
1 - MB - Muito Baixo
0 - NE - Nenhum ou Ausente

Teleprocessamento

Foram testadas a relação de dependência entre o uso de teleprocessamento e os atributos: tipo de setor, tipo de empresas, volume de dados de entrada e de saída, volume de dados armazenados em memória permanente, porte do sistema, grau de interação com o usuário, tempo de resposta, relação com o tempo e o grau de desempenho necessário.

O uso de teleprocessamento (Tabela 4) é mutuamente dependente do setor do mercado que a empresa atua e tem correlação com o

volume de dados de entrada e de saída, sendo a dependência da tecnologia em relação aos atributos, embora a recíproca não seja verdadeira.

O grau de desempenho, o volume de dados permanentes e o uso de teleprocessamento tem uma relação dependente nos dois sentidos e foi comprovado que não existe relação entre o uso dessa tecnologia e o grau de interação com usuário.

Não foi possível determinar a correlação entre os atributos de relação com o tempo e o porte do sistema com o uso dessa tecnologia.

O tempo de execução do sistema é dependente do uso de teleprocessamento, não podendo ser avaliado o inverso.

TABELA 4 - ATRIBUTOS DETERMINANTES DO USO DE TP

	Com TP	Sem TP
Quantidade Sistemas	127	70
SETOR MERCADO: (%)	ST 42 DP 38 QP 37	ST 26 DP 11 QP 33
VOLUME DE DADOS ENTRADA (Mb)	16.877	639
VOLUME DE DADOS SAÍDA (PAG/MES)	147438	12356
VOLUME DE DADOS PERMANENTES (Mb)	1.648	368
QUALIDADE.Desempenho	E/ME	MA/B

Escala de Valores :
- 5 - E - Extremamente Alto
- 4,5 - MA - Muito Alto
- 4 - A - Alto
- 3 - ME - Médio
- 2 - B - Baixo
- 1 - MB - Muito Baixo
- 0 - NE - Nenhum ou Ausente

Setor do Mercado:
- ST - Serviços de Transporte
- DP - Distribuição Petróleo
- QP - Química e Petróleo

Métodos Numéricos

A relação de dependência analisada foi em relação com os seguintes fatores: quantidade de fórmulas, complexidade dos cálculos, grau de confiança e grau de precisão do sistema.

Foi constatado (Tabela 5) que a quantidade de fórmulas tem correlação com o uso de métodos numéricos, embora essa relação seja no sentido que o uso de métodos numéricos influencia na quantidade de fórmulas usadas. A complexidade dos cálculos e a quantidade de transações de cálculos não tem relação de dependência e não pode ser comprovada a relação do uso dessa tecnologia em função do grau de confiança e de precisão necessário ao sistema.

TABELA 5 - ATRIBUTOS DETERMINANTES DE MÉTODOS NUMÉRICOS

Quantidade de Sistemas	Com Met.Num. 11	Sem Met.Num. 186
QUANTIDADE DE FÓRMULAS:	MA/ME	A/NE

Escala de Valores :

> 200	- MA	- Muito Alto
> 70 e ≤ 200	- A	- Alto
> 30 e ≤ 70	- ME	- Médio
> 10 e ≤ 30	- B	- Baixo
> 1 e ≤ 10	- MB	- Muito Baixo
0	- NE	- Nenhuma

5. CONCLUSÃO

No estudo bibliográfico encontrou-se uma grande quantidade de atributos das aplicações, sendo que alguns deles não estão relacionados diretamente com a aplicação ou com sua implementação computacional mas com o ambiente operacional do software, seu usuário e sua empresa. E assim podemos concluir que uma aplicação

de software tem relação com a sua área de atuação, sua empresa, seu usuário e com as técnicas computacionais e matemáticas utilizadas em sua construção.

Os resultados da pesquisa de campo apresentados nesse trabalho são referentes ao universo pesquisado, não podendo ser generalizado para todas essas aplicações em outros ambientes. Embora exista essa limitação, esta pesquisa serviu para fornecer um conhecimento mais profundo sobre as aplicações, seus componentes e suas relações com o ambiente e as tecnologias utilizadas.

Os atributos determinantes ou não das diversas tecnologias (banco de dados, teleprocessamento, métodos numéricos, tipo de processamento, etc.) poderão auxiliar na decisão sobre a adequação ou não do uso de uma determinada tecnologia para um projeto específico, bem como no método de desenvolvimento usado numa determinada fase, pois estes, normalmente, baseiam-se na tecnologia utilizada. Eles forneceram subsídios para a construção de base de conhecimentos da Estação TABA.

Essa pesquisa de campo levou a outros resultados tais como a taxonomia de domínios de aplicação [Werneck 90 e Werneck 90a], conhecimento do processo de desenvolvimento de software utilizado nas empresas, etc.

O processo utilizado, na maioria das empresas, baseia-se em métodos estruturados, principalmente Análise Estruturada segundo o enfoque Cris Gane [Gane 83], Projeto Estruturado [Page-Jones 88] e Modelo de Entidades e Relacionamentos de Chen [Chen 76] com extensões.

Em geral os métodos sofrem adaptações para atender às realidades das empresas. Estas modificações são feitas por pessoal interno das próprias empresas ou externo, através de consultorias, gerando uma documentação aplicativa dos métodos a serem utilizados durante todo o processo de desenvolvimento.

Algumas destas empresas utilizam ferramentas CASE (engenharia de software auxiliada por computador) para o processo de desenvolvimento de diagramas de fluxo de dados e dicionário de dados, bem como para o planejamento e controle de projetos.

O trabalho de campo [Werneck 90] pode ser base para futuras pesquisas através de um estudo em maior profundidade do método de desenvolvimento aplicado e dos resultados obtidos, tendo em conta a necessidade de se avaliar a teoria estudada junto à prática e devido à natureza incipiente da engenharia de software.

O processo de desenvolvimento de software está sendo constantemente estudado e inovado, mas esses estudos nem sempre chegam às empresas, devido à dificuldade do mercado de assimilar novas tecnologias e pela própria inovação, que gera um investimento, normalmente, alto de aprendizagem.

Essa pesquisa possibilitou um contato com a realidade das empresas, dedicando-se a estudar as aplicações, reunindo aspectos encontrados na teoria e na prática. No relatório de perspectivas de pesquisas em engenharia de software [ACM 90], é ressaltada a importância desse tipo de pesquisa que possibilita o contato e o intercâmbio de experiências entre os pesquisadores e os práticos, acarretando na identificação de bons assuntos a serem pesquisados e na validação ou não das soluções propostas através de teorias. Esperamos haver contribuído um pouco para esse fim.

6. BIBLIOGRAFIA

[Acm 1990] "Scaling Up: A Research Agenda for Software Engineering", Communication of the ACM, Vol. 33, No. 3, pp. 281-293, Março de 1990.

[Aguiar 89] Aguiar,T.C.; "Protótipo do Especificador de Ambientes da Estação TABA"; Relatório Técnico ES-208/89 - COPPE/UFRJ; Julho de 1989.

[Blum 86] Blumm, B.I. e Sigillito, V.G.; "An Expert Systema for Designing Information Systems"; John Hopkins APL Technical Digest; Volume 7 no 1; 1986.

[Gane 83] Gane, C. e Sarson, T; "Análise Estrutuada de Sistemas", Livros Técnicos e Científicos Editora, 1983.

[Page-Jones 88] Page-Jones,Meilir; "Projeto Estruturado de Sistemas", Editora McGraw Hill, São Paulo, 1988.

[Rocha 89] Rocha, A.R.C. et all; "O Meta-Ambiente da Estação TABA"; XV Conferencia Latino Americana de Informática; Santiago, Chile; Julho de 1989.

[Rocha 89a] Rocha, A.R.C. et all; "O Meta-Ambiente da Estação TABA"; Relatório Técnico - COPPE/UFRJ; Março de 1989;

[Rocha 90] Rocha, A.R.C.; Souza,J.M.; Aguiar,T.C.; "TABA: A Heuristic Workstation for Software Development"; COMPEURO 90; Tel Aviv, Israel; Maio de 1990.

[SPSS 84] SPSS,Inc.; "SPSS/PC - SPSS for IBM PC/XT", SPSS Inc., USA, 1984.

[WASSERMAN 82] WASSERMAN,A.;"Automated Tools in The Information System Development Environment", IFIP, 1982.

[Werneck 88] Werneck, J. L.; "O Estágio de Desenvolvimento da Área de Planejamento do Setor Textil no Brasil"; tese Msc. COPPEAD/UFRJ; 1988.

[Werneck 90] Werneck, V.M.B.; "Taxonomia de Domínios de Aplicação"; Tese Msc COPPE/UFRJ; Novembro 1990.

[Werneck 90a] Werneck, V.M.B., Aguiar,T.C. e Rocha,A.R.C.; "Classificação de Domínios de Aplicação para Especificação de Ambientes de Desenvolvimento de Software"; CLEI - Congresso Latino-Americano de Informática; Assunção, Paraguai; setembro de 1990;

TRANSFORMACIONES DE PROGRAMAS
DERIVACION DE VERSIONES APLICATIVAS EFICIENTES

Liliana Favre
Facultad de Ciencias Exactas
Universidad Nacional del Centro
de la Pcia. de Buenos Aires
Pinto 399-Tandil-7000
Argentina

Resumen

La construcción formal de programas por transformaciones (programación transformacional) es un método de desarrollo de software que permite derivar un programa desde una especificación formal de un problema a través de pasos de transformación que preservan la correctitud.
Las experiencias realizadas en síntesis de programas mediante programación inferencial muestran que las transformaciones por reglas en un nivel aplicativo son pasos cruciales para lograr versiones eficientes en un nivel procedural. Se presentan en este trabajo un conjunto de reglas de transformación (folding, unfolding, definición, instanciación, abstracción , embedding, reglas sobre los tipos de datos primitivos, reglas sobre el condicional, reglas sobre expresiones lógicas) y estrategias para aplicarlas (composición, generalización, tupling, y embedding funcional) que constituyen las bases para un sistema de transformaciones.

1. Introducción a la programación transformacional

Las diversas metodologías de programación organizan la construcción de un programa a través del desarrollo de una secuencia de versiones. Los criterios que definen las transiciones entre una versión y su sucesora dependen de la metodología empleada.

En las metodologías basadas en **refinamientos sucesivos** ([DIJ76]) las sucesivas versiones incrementan el grado de detalle, sin alterar la estructura exterior de las versiones previas. La verificación de correctitud es totalmente dependiente del problema.

En las metodologías basadas en transformaciones (programación inferencial) se permiten sólo transiciones formales que preservan una relación semántica. Usualmente el proceso comienza con una especificación (formal) y termina con una versión ejecutable. Las transiciones individuales entre varias versiones resultan de aplicar reglas de transformación que preservan la correctitud . La versión final es correcta por construcción. Las transiciones pueden ser descriptas por reglas esquemáticas y pueden ser empleadas en toda clase de problemas. La estructura del programa es flexible, puede ser alterada en el proceso de desarrollo por ejemplo, combinando dos recursiones en una.

Los primeros avances en transformaciones de programas estuvieron asociados a **transformaciones por esquema** de versiones recursivas en versiones iterativas([DAR74],[WAL73]).La aplicación directa de transformaciones esquemáticas limita generalmente las posibilidades de obtener eficientes versiones. Esto motivó la aparición de transformaciones por reglas más flexibles que intentan transformar versiones aplicativas en otras para las cuales existe un tratamiento esquemático eficiente.

Fueron descubiertas las reglas de **folding** y **unfolding** independientemente por Burstall y Darlington([BUR77]) y por Manna y Waldinger [MAN75]. Esta línea de trabajo fue enriquecida con posteriores aportes que precisaron otras reglas de transformación y estrategias asociadas para aplicarlas.

La transformación por folding y unfolding involucra pequeños pasos y es un serio problema controlar el proceso de transformaciones. Requiere del programador la invención de definiciones, bautizadas por Burstall y Darlington como pasos eureka. Son pasos **inteligentes** en el proceso de transformaciones. Pueden compararse con la invención de invariantes de un ciclo en metodologías basadas en refinamiento. Para derivar pasos eureka se emplean **estrategias**. Las mismas juegan el rol de los paradigmas que ayudan al programador a elegir la estructura de un ciclo en refinamientos sucesivos.

Si bien un desarrollo por programación transformacional puede ser hecho con lápiz y papel, existen suficientes razones para usar un sistema implementado para soportar programación inferencial:

- el sistema podría realizar la reescritura de fragmentos de programas e instanciación de ciertas partes de una versión
-el sistema podría asistir al programador con propuestas para aplicar reglas de transformación, como asimismo almacenar la experiencia de los programadores como estrategias de transformación.
-el sistema podría registrar cada decisión de diseño desde la especificación inicial hasta la versión final. Esta historia serviría como documentación, para modificar decisiones de diseño intermedias y para mantenimiento (ante la modificación de la especificación inicial podría repetirse el desarrollo registrado con la especificación modificada).

Existen variados sistemas de transformación que soportan:
-modificación de programas (optimización de estructuras de control, implementación eficiente de estructuras de datos y adaptación de programas a un estilo particular, por ejemplo aplicativo, procedural, etc.)
-síntesis de programas , la generación de un programa desde una descripción formal de un problema
- adaptación de programas a ambientes particulares.

Para un análisis comparativo de sistemas de transformación consultar [PAR84].

Importantes aportes a la programación transformacional ha efectuado el proyecto CIP ("Computer-Aided , Intuition-Guided Programming") dirigido por Bauer. Las actividades fundamentales desarrolladas en este proyecto fueron :

-el diseño y definición formal de un lenguaje de amplio espectro, CIP-L , basado en el ciclo de vida de desarrollo de programas por transformaciones [BAU85]. Este lenguaje permite que coexistan en una misma estructura sintáctica y semántica versiones descriptivas, aplicativas y procedurales.

-el diseño e implementación de un sistema de transformaciones, CIP-S [BAU87]. En principio este sistema podría ser considerado como un sistema experto con una base de conocimiento que es obtenida y extendida por razonamiento formal.

-el desarrollo de una metodología de programación transformacional [PAR90].

Las experiencias realizadas en síntesis de programas mediante programación inferencial muestran que las transformaciones por reglas en un nivel aplicativo son pasos cruciales para derivar versiones procedurales eficientes. Por ejemplo, permiten transformar una doble recursión en una recursión tail, obtener definiciones recursivas menos redundantes, o reemplazar algunos operadores por otros menos costosos. Se presentan en la sección 2. de este trabajo las bases de un sistema de transformaciones de versiones aplicativas. Se describen en 2.1 un conjunto de reglas de transformación y en 2.2 estrategias para aplicarlas. Se ejemplifican en 3. derivaciones realizadas en este contexto.

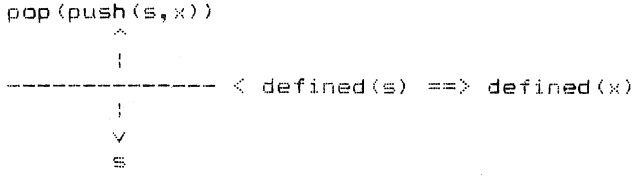
2. Transformaciones por reglas en un nivel aplicativo

(El lenguaje empleado es un subconjunto de CIP-L, el lenguaje aplicativo [BAU85]).

2.1. Reglas de transformación

Reglas generales

Incluyen reglas asociadas a los axiomas de los tipos primitivos bool, char, bit, nat, int, sequ (ver [BAU85]) y reglas originadas en el ambiente de tipos. Por ejemplo, en un ambiente que contiene la definición algebraica de un tipo STACK debería existir el axioma:
pop(push(s,x)) = s para todo s de tipo Stack y todo x de tipo nat
Este axioma se corresponde a la regla



(El predicado DEFINED caracteriza expresiones que no conducen a evaluaciones indefinidas.)

Regla de unfolding

Consiste en efectuar el reemplazo textual de una llamada a una rutina por su cuerpo bajo sustitución textual de los parámetros formales por los objetos concretos (argumentos).

Regla de folding

Consiste en el reemplazo textual de una expresión por una invocación a una rutina cuyo cuerpo es igual a la expresión después de la sustitución de los parámetros.

Regla de instanciación

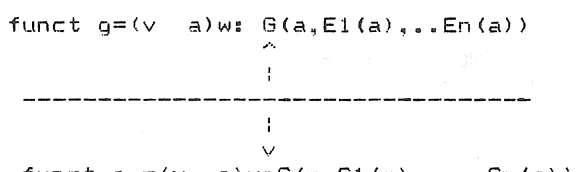
Si en una invocación a una rutina un argumento es constante la operación de unfolding causa que este objeto constante aparezca en el cuerpo de la rutina originando todas las simplificaciones posibles, a esto se lo denomina **instanciación parcial** .

Regla de definición

Introduce una nueva rutina cuyo cuerpo no puede obtenerse mediante la instanciación parcial de una rutina existente.

Regla de abstracción

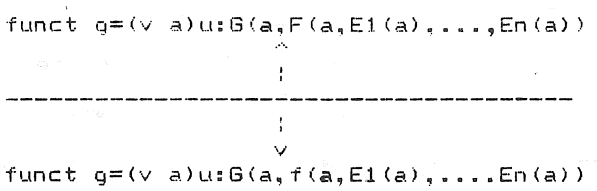
Sea una rutina g con diferentes subexpresiones Ei(a) de modos mi que están contenidas en una expresión G de modo w, vale para g la siguiente transformación:



La transformación es válida en ambas direcciones. La equivalencia está basada en el principio de sustitución([BAU82]).

Regla de embedding

Una rutina g con diferentes subexpresiones Ei(a) de modos mi(i=1,..n) que están contenidas en una expresión F de modo w, puede transformarse , bajo ciertas condiciones, mediante la siguiente transformación:

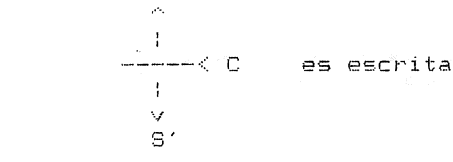


La nueva rutina f tiene un parámetro adicional para cada una de las subexpresiones a ser tratadas. El cuerpo de f es obtenido reemplazando cada ocurrencia Ei(a) en F por el correspondiente parámetro si. Esta forma general de la regla describe un embedding incompleto. Si la expresión encerrada es el cuerpo entero de la rutina original, el embedding es completo.

Esta transformación es válida si la terminación de g implica la terminación del sistema (g,f). Es suficiente pero no necesario que g no sea recursiva, es decir no sea invocada en alguna de las expresiones Ei(a). Además, todas las llamadas de F deben estar definidas. La transformación es válida en el sentido opuesto si cada Ei(a) está determinada([BAU82]).

Reglas asociadas a expresiones lógicas

Las reglas son escritas en notación lineal, así



El predicado DETERMINATE caracteriza expresiones determinadas. El predicado DEFINED caracteriza expresiones que no conducen a evaluaciones indefinidas.

1. (P cand Q) cand P <---> P cand Q >--< determinate(P)

2. (P and Q) cand R<-->P cand (Q cand R) >-<(wp(P)=F)==>defined(Q)

3. (P cand Q) cand R <--> P cand (Q cand R)

4. (P and Q) cand R <--> P and (Q cand R)
   >--< (wp(P) =F and wp(Q) = T) ==> defined(R)

5. P and (Q cand R) <--> (P and Q) cand R
   >--< (wp(P)= F and wp(Q)= T) ==> defined(R)

6. (P cand Q) and R<-->P cand (Q and R)>-<(wp(P)=F)==>defined(R)

7. (P cand Q) cand R <--> P cand (Q cand R)

8. (P and Q) and (R and S) <--> (P and R) and (Q and S)

9. P cand Q <--> Q and P >--< (wp(P)= F) ==> defined(Q)

10. P cand (Q and R) <--> Q and (P cand R)>--<wp(P)==> defined(Q)

11. P and Q <--> P cand Q >--<(wp(P)= F) ==> defined(Q)

12. P ==> Q <--> P ==> Q >--< (wp(P)= F) ==> defined(Q)

13. P OR Q <--> P cor Q >--< (wp(P) = T) ==> defined(Q)

14. P cand Q <--> P and Q >--< (wp(P)=F) ==> defined(Q)

15. P <==> Q <--> (P ==> Q) and (Q ==> P)

16. P and Q <--> Q and P

17. P <==> Q <--> Q <==> P

18. P or Q <--> Q or P

19. P cand Q <--> Q cand P
    >--<(wp(P)=F)==>defined(Q)) and ((wp(Q)=F)==>defined(P))

20. P cand (Q cand R) <--> (Q cand (P cand R)
>--<((wp(P)=F)==>defined(Q)) and ((wp(Q)=F)==>defined(P))

21. P and P <--> P

22. P or P <--> P

23. P cor P <--> P

24. P cand (Q and R) <--> (P cand Q) and (P cand R)

25. P cand (Q or R) <--> (P cand Q) or (P cand R)
>--< determinate(P)

26. P cor (Q cand R) <--> (P cor Q) cand (P cor R)
>--< determinate(P)

27. (P and Q) and R <--> (P and R) and (Q and R)

28. (P and Q) and R <--> (P and (Q and R))

29.P and false <--> false >--< defined(P)

30.P and true <--> P

31.true and P <--> P

32.P <==> P >--< defined(P) and determinate(P)

33.P <==> false <--> not P

34. false and P <--> false >--< defined(P)

35.false or P <--> P

36.false cand P <--> false

37. false ===> P <--> true

38.not false <--> true

39. P or false <--> P

40.P cand false <--> false >--< defined(P)

41.P cand true <--> P

42.true and P <--> P

43.true cand P <---> P

44.(P<==>Q) cand R <--> (P <==> Q) cand S

45. P and not P = false

Reglas del condicional

1.

```
 if A then if B then E1
                   else E2
           fi
      else
           if B then E3
                   else E4
           fi
  fi
```

```
 if B then if A then E1
                   else E3
           fi
      else if A then E2
                   else E4
           fi
  fi
```

2.

```
  if A and B              then E1
  #  A and (not B)        then E2
  # (not A) and C         then E3
  # (not A) and (not C)   then E4
  fi
```

```
  true)          ¦ (A,B,C determinada
                 v (A,B,C definidas
```

```
  if A  then  if  B  then  E1
                           else  E2
                 fi
         else  if  C  then  E3
                           else  E4
                 fi
    fi
```

3.

```
 F(if C then B1 else B2 fi)
```

```
 if C then F(B1)
      else F(B2) fi
```

4.

```
 F( if C then  E  fi)
```

```
   if C  then  F(E) fi
```

5.

```
 F(if A then B else C fi, D)
```

```
 if A then F(B,D)
      else F(C,D) fi
```

6.

```
  F( if C  then  E fi,P2,P3)
```

```
   if C then F(E,P2,P3) fi
```

7.

```
 F(D,if A then B else C fi)
```

```
 if A then F(D,B)
      else F(D,C) fi
```

8.

```
 if  true  then  E   fi
```

```
 E
```

9.

```
 if false then E else F fi
              ^
              |
 -------------------------
              |
              v
              F
```

10.

```
       if true  then E else F fi
                  ^
                  |
      ------------------------
                  |
                  v
                  E
```

11.

```
 if C then E else E fi
              ^
              |
 ---------------------
              | ( defined(C)
              v
              E
```

12.

```
       if A  then E1 else E2 fi
                  ^
                  |
      ------------------------
                  |
                  v
       if not A then E2 else E1 fi
```

13.

```
if A then if B then E1
                 else E2
             fi
     else E3
 fi
              ^
              |
 ------------------------
              |(wp(A)==>B   |
              |
              v
if A then E1 else E3 fi
```

14.

```
 if  A  then  E1
        else  if  B  then  E2
                           else  E3
                  fi
  fi
                   ^
                   |
  ---------------------------
                  |((wp(A)=F)==>
                  |       not B
                  v
    if A  then  E1  else E3 fi
```

15.

```
if C then if D then E1
                 else E2
           fi
fi
              ^
              |
------------------------
              |
              v
if D  then if C then E1
              fi
      else if C then E2
              fi
 fi
```

16.

```
 if A  then G(F(E1,E3))
       else G(F(E2,E3))
 fi
              ^
              |
 ------------------------
              |
              v
   G(F(if A then E1 else E2 fi,E3))
```

17.

```
F(D,if A then B else C fi,E)
              ^
              |
------------------------------
              |
              v
 if A then  F(D,B,E)
      else  F(D,C,E)
 fi
```

18.

```
if A  then x
     #  B  then y
     else  x
     fi
          ^
          |
          ------------  { defined(A)
          | and wp(B)==> not A
          v
     if B then y else x fi
```

19.

```
 if C1 then   E1
 #  C2 then   E2
 ...................
 #  Ci   then Ei
 #  Ci+1 then Ei+1
 ...................
 fi

         ^
         |
 ---------------------
         | {valor(Ei)=F
         v

 if C1   then E1
 #  C2   then E2
 ...................
 #  Ci+1 then Ei+1
 ...................
 fi
```

20.

```
           if C1  then  E1
          #  C2  then  E2
          ...............
          #  Ci  then  Ei
          ...............
          #  Cj  then  Ei

          fi
                ^
                |
          ---------------------
                |

                v
          if C1      then  E1
          #  C2      then  E2
          ...................
          #  Ci OR Cj then Ei
          ...................
```

21.

```
          if A then E1
               else if B then E2
                         else if C then E3
                              #  D then E4
                              else  E5
                              fi
                    fi
          fi
                         ^
                         |
          ---------------------------------------{ A,B determinada
                         |
                         v
          if A  then E1
               else if B then E2
                         else if not A and not B and C then E3
                              #  not A and not B and D then E4
                              else E5
                              fi
                    fi
          fi
```

2.2. Estrategias para aplicar reglas de transformación

Estrategia de composición

Si una expresión f(g(x)) de modo w ocurre en un sistema de rutinas es posible definir una nueva rutina h:

funct h=(m x)w:f(g(x)).

y reemplazar las ocurrencias de f(g(x)) por h(x) en el sistema.

Estrategia de generalización

Si una expresión E(x,c) (x variable libre y c constante) de modo w ocurre en una rutina de un sistema es posible definir una nueva rutina:

funct f=(mx x,my y)w: E(x,y).

y reemplazar las ocurrencias de E(x,c) en el sistema por f(x,c).

Estrategia tupling

Si dos funciones f(x) y g(x) requieren la evaluación de la función h(x), es conveniente definir una nueva función

p(x)=<f(x),g(x)>

y usar la regla de abstracción en p(x) para calcular h(x) una sóla vez.

Estrategia de embedding funcional

Dada una rutina A es posible mediante la estrategia de generalización, definir una nueva rutina B tal que una instanciación parcial de B inmersa en otra rutina C defina una equivalencia de C con A.

3. Ejemplos

Se presentan a continuación ejemplos que pretenden mostrar el uso combinado de estrategias para aplicar reglas de transformación El Ejemplo 3 muestra que la aplicación del método planteado permite obtener versiones aplicativas eficientes en tiempo partiendo de versiones de complejidad temporal exponencial.

Ejemplo 1

Sea

```
1.    funct sqsum=( nat n) nat:
      if n=0 then  0
      else sq(n) + sqsum(n-1)
      fi.
```

2.Regla de definición- Estrategia de generalización

```
      funct gsqsum=( nat n, nat nsq)nat:
      if n=0  then  0
              else  nsq + gsqsum(n-1,nsq-2*n+1)
      fi.
```

3. Estrategia de embedding funcional

1. es equivalente al sistema:

```
     funct sqsum=(nat n)nat: gsqsum(n,sq(n))
     where
     funct gsqsum=(nat n, nat nsq)nat:
     if n=0  then  0
             else  nsq + gsqsum(n-1,nsq-2*n+1)
     fi.
```

Ejemplo 2

Derivación de un algoritmo que calcule la suma de los cuadrados de una secuencia de reales

Sea la definición de un tipo secuencia:

```
type  SEQU  = (sort m) sequ,  empty,  isempy,  make ,&  ,  top  ,
rest ,append ,bottom, upper ,stock ,length ,select ,change:
sort sequ,
sequ empty,
funct (sequ) bool isempty,
funct (m) sequ make,
funct (sequ,sequ) sequ ,&,
funct (sequ s:not isempty(s)) m top,
funct (sequ s:not isempty(s)) sequ rest,
funct (sequ, m) sequ append,
funct (sequ s: not isempty(s)) sequ upper,
funct (sequ, m) sequ stock,
funct (sequ) nat length,
funct (sequ s, nat i:1<=i and i<=length(s)) m change;
funct (sequ s: not isempty(s)) m bottom;
funct (sequ s,nat i:1<=i and i<=length(s)) m select;
laws m x, sequ r, sequ s, sequ t:
1.      (r & (s & t)) = ((r & s) & t)
2.      (s & empty) = s, (empty & s)= s,
3.      isempty(empty) = true,
4.      isempty(make(x)) = false,
5.      isempty(s & t) = (isempty(s) and isempty(t)),
6.      append(s,x) =(make(x) & s),
```

```
7.      top(append(s,x)) = x,
8.      rest(append(s,x)) = s,
9.      stock(s,x) = (s & make(x)),
10.     bottom (stock(s,x)) = x,
11.     upper(stock(s,x)) = s
12.     length(empty) = 0
13.     length(make(x)) = 1,
14.     length(s & t) = length(s) + length(t)
15.     (i <= length(s)) ==> select(s,i) = top(s),
16.    (1 < i and i<=length(s)) ==> select(s,i) =select(rest(s),i-1)
17.    (i <= length(s)) ==> change(s,1,x)=append(rest(s),x),
18.    (1 < i and i<=length(s) ) ==>
       change(s,i,x)=append(change(rest(s),i-1,x),top(s))
end of type.
```

Sean

1.

```
funct cuadrado=(sequ real L)sequ real:
if L=@  then   @
        else top(L)*top(L) & cuadrado(rest(L))
fi.
```

2.

```
funct suma=(sequ real L)real:
if  L=@   then  0
          else  top(L) + suma(rest(L))
fi.
```

3. Definición- Estrategia de composición

```
funct sumcuad=(sequ real L)real:
suma(cuadrado(L)).
```

4. Unfolding en 3. con 2.

```
funct sumcuad=(sequ real L)real:
if  L=@   then   0
          else   top(cuadrado(L)) + suma(rest(cuadrado(L)))
fi.
```

5. Unfolding en 4. con 1.

```
funct sumcuad=(sequ real L)real:
if  L=@  then  0
         else
        5'. top(if L=@  then  @
                else top(L)*top(L)&cuadrado(rest(L))
                fi)  +
        5''. suma(rest(if L=@  then  @
                       else  top(L)*top(L)&cuadrado(rest(L))
                       fi)
fi.
```

6.Regla del condicional 16 en 5'.y 5''. :

```
funct sumcuad=(sequ real L)real:
if L=@  then 0
        else  (if  L=@  then top(@)
                        else  top(top(L)*top(L)&cuadrado(rest(L)))
               fi)  +
        (if  L=@  then suma(rest(@))
                  else suma(rest(top(L)*top(L)&cuadrado(rest(L))))
         fi)
 fi.
```

7. Por reglas generales, axiomas de sequ 7 y 8

```
funct sumcuad=(sequ real L)real:
if L=@  then  0
        else
            (if L=@  then top(@)
                     else top(L)*top(L)
             fi) +
            (if L=@  then suma(rest(@))
                     else suma(cuadrado(rest(L))
             fi)
fi.
```

8. Regla del condicional 16

```
funct sumcuad=(sequ real L)real:
if  L=@  then  if  L=@  then  0
                       else top(@) +
                       (if L=@  then suma(rest(@))
                                else  suma(cuadrado(rest(L))
                        fi)
            fi
     else  if L=@  then  0
                   else top(L)*top(L) +
                        (if L=@  then suma(rest(@))
                                 else suma(cuadrado(rest(L))
                         fi)
            fi
fi.
```

9. Por regla del condicional 2:

```
funct sumcuad=(sequ real L)real:
if L=@ and L=@       then  0
#  L=@ and not(L=@)  then  top(@) + (if L=@  then  suma(rest(@))
                                     else suma(cuadrado(rest(L)) fi)
#  not(L=@) and L=@ then  0
#  not(L=@) and not(L=@) then  top(L) *top(L) +
                               (if L=@ then suma(rest(@))
                                       else suma(cuadrado(rest(L)) fi)
fi.
```

10.
L = @ and L=@ --> L = @ por regla 21 de expresiones lógicas,

L = @ and not(L = @) --> false por regla 45 de expresiones lógicas,

not(L=@) and L = @ ---> L=@ and not(L=@) --> false por regla 16 y 45 de expresiones lógicas,

not(L=@) and not(L=@) ---> not(L=@) por regla 21. de expresiones lógicas

```
funct sumcuad=(sequ real L)real:
if  L=@     then  0
#   false   then  top(@) +
                  (if L=@  then suma(rest(@))
                           else suma(cuadrado(rest(L))
                   fi)
# false     then  0
# not(L=@)  then  top(L)*top(L) + (if L=@ then suma(rest(@))
                                   else suma(cuadrado(rest(L))
                                   fi)
fi.
```

11. Por regla 19. del condicional

```
funct sumcuad=(sequ real L)real:
if  L=@     then  0
# not(L=@)  then  top(L)*top(L) + (if L=@ then suma(rest(@))
                                   else suma(cuadrado(rest(L))
                                   fi)
fi.
```

12. Regla 7. del condicional

```
funct sumcuad=(sequ real L)real:
if  L=@     then  0
# not(L=@)  then  if  L=@ then top(L)*top(L) + suma(rest(@))
                  else top(L)*top(L) + suma(cuadrado(rest(L))
                  fi
fi.
```

13. Regla 14 del condicional

```
funct sumcuad=(sequ real L)real:
if L=@      then  0
# not(L=@)  then  top(L)*top(L) + suma(cuadrado(rest(L))
fi.
```

14. Folding de 13. con 3.

```
funct sumcuad=(sequ real L)real:
if  L=@      then  0
# not (L=@) then  top(L)*top(L) + sumcuad(rest(L))
fi.
```

Esta última versión evita generar una secuencia intermedia de cuadrados como la versión inicial.

Ejemplo 3

El problema de las curvas de Hilbert

Sean
movimentos={N,S,E,O} un conjunto de posibles movimientos (hacia el norte, hacia el sur, hacia el este y oeste respectivamente)

```
mode movi=atomic (N,S,E,O)
mode secuencia=(sequ movi w,sequ movi x,sequ movi y,sequ movi z)
```

y el siguiente sistema de rutinas

1.
```
funct Hilbert(nat n)sequ movi:
A(n).
```

2.
```
funct A=(nat n) sequ movi:
if  n=0  then @
         else D(n-1) & O & A(n-1) & S & A(n-1) & E & B(n-1)
fi.
```

3.
```
funct B=(nat n) sequ movi:
if  n=0  then @
         else C(n-1) & N & B(n-1) & E & B(n-1) & S & A(n-1)
fi.
```

4.
```
funct C=(nat n) sequ movi:
if  n=0  then @
         else B(n-1) & E & C(n-1) & N & C(n-1) & O & D(n-1)
fi.
```

5.
```
funct D=(nat n) sequ movi:
if  n=0  then @
         else A(n-1) & S & D(n-1) & O & D(n-1) & N & C(n-1)
fi.
```

Las funciones A,B,C,D muestran claramente evaluaciones repetidas. Esto sugiere aplicar la estrategia tupling. A fin de encontrar una tupla EUREKA se construye el DAG de evaluación simbólica surgiendo la siguiente DEFINICION EUREKA- ESTRATEGIA TUPLING

6.
```
funct T=(nat n)secuencia:
if n= 0  then  <@,@,@,@>
else  < A(n),B(n),C(n),D(n) >
fi.
```

7. Unfolding de 6. con 2.,3.,4.,5.
```
funct T=(nat n)secuencia:
if  n=0  then <@,@,@,@>
         else <D(n-1) & 0 & A(n-1) & S & A(n-1) & E & B(n-1),
               C(n-1) & N & B(n-1) & E & B(n-1) & S & A(n-1),
               B(n-1) & E & C(n-1) & N & C(n-1) & 0 & D(n-1),
               A(n-1) & S & D(n-1) & 0 & D(n-1) & N & C(n-1)>
fi.
```

8. Por abstracción de 7. con 6.

```
funct T=(nat n)secuencia:
if  n=0   then   <@,@,@,@>
          else   f1(T(n-1))
fi where
funct f1=(secuencia u)secuencia:
<u.z & 0 & u.w & S & u.w & E & u.x,
 u.y & N & u.x & E & u.x & S & u.w,
 u.x & E & u.y & N & u.y & 0 & u.z,
 u.w & S & u.z & 0 & u.z & N & u.y>.
```

9. De 1. y 6.

```
funct Hilbert=(nat n)sequ movi:
T(n).w.
```

10. Abstracción en 9. con 6.

```
funct Hilbert=(nat n)sequ movi:
T(n).w where
funct T=(nat k)secuencia:
if  k=0  then  <@,@,@,@>
         else  f1(T(k-1))
fi. where
funct f1=(secuencia u)secuencia:
<u.z & 0 & u.w & S & u.w & E & u.x,
 u.y & N & u.x & E & u.x & S & u.w,
 u.x & E & u.y & N & u.y & 0 & u.z,
 u.w & S & u.z & 0 & u.z & N & u.y>
```

La complejidad temporal de Hilbert es O(n), considerando constantes a las operaciones asociadas a la estructura computacional primitiva sequ.

4. Conclusiones

Los ejemplos derivados muestran que programas eficientes pueden ser derivados en una forma simple, transparente y correcta mediante transformaciones. Por ejemplo, se conocen soluciones recursivas exponenciales en tiempo e intrincadas versiones iterativas sin prueba de correctitud para el problema de las curvas de Hilbert [SPA71]. La versión aplicativa derivada en este trabajo puede transformarse directamente por esquemas en una versión iterativa eficiente de complejidad temporal lineal.

A fin de profundizar el análisis de transformaciones en un nivel aplicativo se requiere de un sistema implementado que soporte la metodología planteada. Se continúa en esta línea de trabajo.

5. Referencias bibliográficas

[BAU82] "Algorithmic Language and Program Development" Bauer, F., Wossner, H. Springer Verlag, 1982.

[BAU85] "The Munich Project CIP, vol I : The Wide Spectrum Language CIP-L". Bauer, F. Lecture Notes in Computer Science 183, Springer, Berlin, 1985.

[BAU87] "The Munich Project CIP, vol II : The Program Transformation System CIP-S" Bauer, F. Lecture Notes in Computer Science 292, Springer, Berlin, 1987.

[BAU89] "Formal program construction by Transformations--Computer-Aided, Intuition Guided Programming", Bauer, F.L., Moller, B., Partsch, H., Pepper, P. IEEE Transactions On Software Engineering, Vol 15. NO. 2, Febraury 1989.

[BOI90] "USTOPIA Requirements Thoughts on a User-friendly System for Transformation of Programs In Abstracto". Boiten, E.A., van der Brand, M.G.J., van Diepen, N.W.P.,Koster, C.H.A. Technical Report NO. 90-12,June 1990. University of Niejmegen. Department of Informatics.

[BUR77] "Program Development by Transformations : an overview". Burstall R. Feather, M. Les Fondements de la Programation(eds. M.Amiochaby e D.Neel) Touluse, Dec 1977

[BUR77-1]"Recursive programs:proof, transformation and synthesis" Burstall,R. Rivista di Informatica, vol VII, suplemento al, n. 1, gennaio-marzo 1977.

[BUR77-2]" A Transformation System for Developing Recursive Programs". Burstall, R.;Darlington, J. ACM, vol 24,N 1, January 1977.

[DAR74] " A System which automatically improves programs." Proc. Sixth Annual ACM Symp. Theory of Comtg.,1974, pp 13-26.

[DIJ76] " A Discipline of Programming". Dijkstra, E. W. Englewood Cliffs, N.J. :Prentice Hall 1976.

[MAN75] "Knowledge and reasoning in program synthesis. Manna, Z. and Waldinger, R. Artif. Intel. J.6.2(1975), 175-208.

[PAR83] "Program Transformation systems".Partsch, H.,Steinbruggen, R. Computing Surveys, Vol 15, No. 3,September 1983.

[PAR86] " Program transformations expressed by algebraic type manipulations". Technique et Science Informatiques, vol 5, No. 3. 1986.

[PAR88] " A formal derivation of Boyer and Moore's pattern matching algorithm". Partsch, H., Stomp, F. Internal Report no. 88-12, 1988 department of Informatics. Faculty of Science. University of Nijmegen.

[PAR89] " From informal requirements to a running program: A case study in algebraic specification and transformational programming". Partsch, H.Science of Computer Programming 11(1988/1989) 263-297. North-Holland.

[PAR90] " Specification and Transformation of Programs. A Formal Approach to Software Development". Springer-Verlag. 1990.

[PET82]" Deriving very Efficient Algorithms for Evaluation Linear Recurrence Relations Using the Program Transformation Technique". Pettorossi,A.Burstall,R.Acta Informatica 18,181-206(1982).

[PET84]"A Powerful Strategy for Deriving Efficient Programs by Transformation".
Pettorossi,A. Papers presented at the Symposium 1984 ACM Symposium on Lisp and Functional Programming.pp 273-281.

[SPA71] " Space Filling Curves, or How to Waste time on a Plotter". Softw. Pract. And Exper. Vol 1, No. 4(1971)403-410

[WAL73] "Characterizations of Flowchartable Recursions". Walker,S.A. and Strong, H.R. JCSS Vol. 7 no. 4 (Aug 73) pag 404-447.

A D I

AMBIENTE DE DESARROLLO DE APLICACIONES PARA PROCESAMIENTO DE IMAGENES DIGITALES.

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Querétaro

Autores :

Idalia C. Solis Castro.
Ma. Luisa Herrera López.
Edgar Gerardo Mondrágon Martinez.

Alumnos de 9º semestre de la carrera de Ingeniería en Sistemas Computacionales, en el ITESM CQ.

Coordinación del Proyecto :

Ing. Carlos E. Romero Ortiz.
Profesor Depto. Sistemas Computacionales

Ing. Salvador Galindo Hernández.
Director del Depto. de Electrónica y Comunicaciones.

Dirección :

Departamento de sistemas Computacionales del
ITESM Campús Querétaro:
Henry Ford No. 10 Parques Industriales
CP 76130 Querétaro, Qro.
México.
Tel. (42) 12 60 90 Ext 160.
Fax: (42) 12 82 50
Bitnet : HMORELOS at VMTECQRO

A D I
AMBIENTE DE DESARROLLO DE APLICACIONES PARA PROCESAMIENTO DE IMAGENES DIGITALES.

RESUMEN

El presente trabajo consiste en el desarollo de una herramienta para generación de aplicaciones en el área de procesamiento de imágenes digitales. La idea del proyecto es el poder contar con una herramienta de alto nivel para el desarrollo de aplicaciones en el área de tratamiento de imagenes digitales.

El término de procesamiento o tratamiento de una imagen se refiere a la utilización de ciertos algoritmos y herramientas para cambiar la apariencia de la imagen original, utilizando para esto información de la misma, o realizando una transformación y aplicando los algoritmos o funciones matemáticas sobre la transformada de la imagen.

Actualmente muchas áreas en la industria y medicina dependen de algún tipo de imágen para diagnòstico o toma de decisiones en ciertos procesos. Estas van desde fotografía normal hasta radiografías, termografías y tomografías. La calidad de la imágen es un factor de grán importancia y en algunos casos es complicado lograr algo óptimo. Las técnicas antes mencionadas dan una alternativa hacia éste problema, el poder tratar de cierta forma la imágen y lograr resaltar o hacer más claros ciertos objetos o áreas de interes por medio de una computadora. Esto implica beneficios en cuanto al costo del proceso, debido a que en algunos casos el material para volver a tomar la imágen son demasiado caros o por la situación es casi imposible volver a hacerlo.

La herramienta que se está desarrollando provee un puente de alto nivel entre los lenguajes de programación convencionales y el usuario o desarrollador de la aplicación en una área específica. Una de las partes importantes del proyecto es la inclusión de un lenguaje orientado a objetos con características de alto nivel. Esto permitirá que las aplicaciones utilizando las técnicas de tratamiento de imágenes sean más comunes y aprovecharlas en distintos campos de México y América Latina.

Esto es como parte de los proyectos del Departamento de Sistemas Computacionales del ITESM Campús Querétaro.

DESARROLLO

INTRODUCCION

El uso de técnicas de procesamiento nos permite utilizar de una manera más completa la información que pudiera contener una imagen tomada de alguna forma (fotografía normal, termografía, radiografía, tomografía, etc.). AL tener una mejor información con respecto a la imagen, podemos hacer un uso más eficiente de los recursos, ya sea tiempo para tomar una fotografía mejor, disponibilidad de la fotografía, y de alguna manera mejorar los resultados que se obtendrían con la imagen original.

El término de procesamiento o tratamiento de una imagen se refiere a la utilización de ciertos algoritmos y herramientas para cambiar la apariencia de la imagen original, utilizando para esto información de la misma, o realizando una transformación y aplicando los algoritmos o funciones matemáticas sobre la transformada de la imagen.

La información que tenemos de la imagen recién digitalizada y sobre la que se trabaja inicialmente, es de intensidad luminosa. Si consideramos que se encuentra discretizada en un arreglo de **i** renglones y **j** columnas, tenemos una función de mapeo **f(i,j)** que nos representa la intensidad luminosa o nivel de gris en el punto **(i,j)** de la imagen. Esta intensidad luminosa es discreta y generalmente por facilidad de almacenamiento es un número potencia de 2.

Hay ciertos casos en que la información tal cual de la imágen está mezclada con ruido de tal manera que la combinación de técnicas espaciales no es ideal para mejorarla. Para esto se utiliza la información de frecuencia de la imágen, obteniendola por medio de la Transformada de Fourier. Este espectro de frecuencia puede filtrarse y obtener resultados mejores que en espacial.

HERRAMIENTA DE DESARROLLO

El uso de éstas técnicas permite que muchos campos, donde las imágenes son importantes, se vean beneficiados en la calidad final de sus procesos.
El problema de introducir tecnología nueva es la complejidad de ésta para aprovecharla al máximo. La búsqueda de la facilidad de uso utilizando al máximo el poder de la computadora fué la razón de crear un ambiente completo para el desarrollo de éstas aplicaciones para procesamiento de imagenes digitales. Este provee un puente entre los usuarios o desarrolladores y un sistema sofisticado de tratamiento.

La herramienta fue dividida en varias partes relacionadas, con las cuales se trabaja de forma simultánea :

I. lenguaje orientado a objetos
II. rutinas de soporte para el S.O.(Memoria, E/S).
III. funciones para procesamiento espacial
IV. funciones para procesamiento en frecuencia (Fourier).
V. funciones de soporte para visualización, impresión, compactación y comunicación con dispositivos.
VI. interfase del ambiente final.

Estructuralmente la herramienta se divide en tres partes principales :

◊ Ambiente Externo.
◊ Ambiente Interno.
◊ Ambiente de Usuario (Desarrollo).

Ambiente Externo :
Incluye todas las declaraciones y el codigo objeto de las funciones de soporte y procesamiento. Este codigo es la base para la aplicación final, éstas rutinas van totalmente relacionadas con las declaraciones de objetos del lenguaje del ambiente.
Las rutinas para dispositivos consideradas en esta parte por grupos son:

1) De digitalización.
Las rutinas para estos dispositivos incluyen la toma interactiva de imágenes. El equipo con que se cuenta para esto consiste en :
◊ Cámara para digitalización WINCAM.
◊ Tarjeta para digitalización de video.

2) De entrada interactiva.
Estos dispositivos permiten una comunicación con el usuario de forma interactiva, actualmente se soportan sólo dos :
◊ teclado.
◊ ratón.
La interfase en el lenguaje se hace por medio de mensajes por parte de los dispositivos a la aplicación.

3) De salida impresa.
Estos dispositivos son basicamente impresoras, diferenciandolas en dos tipos principales :
◊ Matriz de puntos.
◊ Impresora Laser.
Para el manejo de la última, se ésta desarrollando un módulo de comunicación al equipo Apple Macintosh para aprovechar completamente la capacidad gráfica del equipo con que se cuenta. Este módulo incluye compactación, decompactación y manejo de comunicación por puerto serial.

4) De salida para visualización.
Estas rutinas dan soporte para la visualización en video de la imágen a procesar y la procesada. Se soporta los manejadores de video para los subsistemas MCGA y VGA, además de manejo de un monitor externo.
Las imágenes con las cuales se ésta trabajando actualmente tienen una resolución hasta de 220 por 256 pixels con 256 niveles de gris.

5) De salida a almacenamiento.
Estas rutinas tienen como función el almacenamiento en disco de las imágenes, así como su lectura y compactación. Para estas tareas se utilizan formatos generales, para asegurar compatibilidad con otros equipos y sistemas de software. Los formatos utilizados incluyen : TIFF, BUF y PCX.

6) De salida a comunicaciones.
La función de éste grupo de rutinas es la de proveer una salida del sistema hacia otras máquinas, incluyendo comunicación con equipos de procesamiento mayores (mainframes). Esto se incluye debido a ciertos paquetes de procesamiento para imagenes muy grandes que existen para mainframes.

I. LENGUAJE ORIENTADO A OBJETOS

La idea de proveer un lenguaje orientado a objetos es la facilidad que dá este tipo de programación en cuanto a visualizar como objetos independientes y completos a los distintos dispositivos y entidades logicas, en lugar de manipular directamente un conjunto de datos y funciones como ocurre en la programación normal.
Esto le dará al usuario un ambiente amigable y sencillo sin restar sofisticación y calidad a las aplicaciones finales que se deseen realizar.

El desarrollo del lenguaje se dividió en las siguientes partes :

- ◊ Especificación de los alcances y capacidad del lenguaje.
- ◊ Evaluación y Elección del lenguaje para la aplicación final.
- ◊ Diseño de la gramática.
- ◊ Elaboración de la tabla de parse.
- ◊ Elaboración del manejador de la tabla (chequeo de sintaxis, chequeo de tipos y referencias a objetos).
- ◊ Elaboración del Generador de Codigo Final.

La idea principal del lenguaje es la de dar al usuario una manera más sencilla de referirse a los dispositivos y a la imágen en sí para procesarla. La base del diseño del lenguaje es la programación orientada a objetos, de acuerdo a esta filosofía de programación, tendremos ahora representados a cada una de las entidades lógicas y físicas por medio de objetos independientes y relacionados con los demás que comparten el medio. La definición para un objeto básico de una imágen sería la siguiente :

```
imagen = objeto
metodos
        invierte.
        desplaza ( x, y ).
        escala ( x, y ).
        ...
atributos
        info (resx,resy).
        resx.
        resy.
        grises.
fin
```

Donde los métodos son las acciones que un objeto dado, en este caso una imágen, puede llevar a cabo consigo misma, y los atributos son las caracteristicas que la definen. Esta definición es para una **clase** general de imágen. Una **instancia** sería la definición específica para una imágen con valores conocidos, un ejemplo de instancia es el siguiente :

```
i1 : imagen ( resx = 100.
              resy = 100.
              grises = 256.
            ).
```

Aqui el objeto i1 está definido como imágen con una resolución de 100 puntos horizontales por 100 verticales y 256 niveles de muestra.
Para usarlo como imágen en sí faltaría darle los datos sobre sí misma, la información en sí de la imágen. En éste punto entra la interacción con los demás objetos definidos. Con los objetos le damos al desarrollador una visión no de datos aislados, sino información interrelacionada y con capacidad de manipularse.

El ambiente al principio contempla una serie de definiciones básicas de objetos, a partir de los cuales se pueden crear subclases con nuevos métodos y atributos para formar una biblioteca y permitir que el desarrollador haga crecer el sistema. En sí estos objetos con métodos y características nuevas son representaciones de las funciones y el código externo definido que se habrá de ligar con el código generado para la aplicación final.

La comunicación entre los objetos puede definirse como un mapeo entre unidades lógicas y dispositivos físicos. Las unidades lógicas definidas inicialmente son :

- ◊ imágen.
- ◊ archivo.
- ◊ Path.
- ◊ histograma.
- ◊ Cada modo de video soportado.
- ◊ Nombre de cada dispositivo externo de digitalización.

Las unidades físicas iniciales son las siguientes:

- ◊ de digitalización.
- ◊ de entrada interactiva.
- ◊ de entrada/salida para almacenamiento.
- ◊ de despliegue.

El mapeo entre unidades lógicas y fisicas o de lógicas a lógicas define las acciones o información que se puede compartir entre ellas. Para una imágen el mapeo sería el siguiente :

- ◊ imágen <-> archivo. : Mapeamos un archivo a una imágen o viceversa.
- ◊ imágen -> despliegue. : Una imágen puede ser mandada a un dispositivo físico de despliegue.
- ◊ imágen <- digitalizador. : Una imágen puede ser digitalizada.

Aqui el mapa de relaciones puede hacerse más complejo al definir objetos nuevos, el compilador antes de generar codigo se encargará de checar y resolver éstas relaciones para evitar conflictos de tipos en la aplicación final.

Los objetos definidos por el usuario incluiran un objeto tipo programa con código para manejo de instancias. El conjunto de éstos programas es el que definirá la aplicación final. Un programa que lea una imágen , invierta sus niveles y la despliegue sería :

```
p : programa (

      i : imagen (   xres=100.
                     yres=100.
                     grises = 256.
                 ).
      d  : camara.
      v  : monitor.

      camara.digitaliza(i).
      i.invierte.
      v.despliega(i).
 ).
```

A partir de éste objeto programa, el compilador checa los errores de sintaxis , de tipos y de referencias, para generar una aplicación final en algún lenguaje de alto nivel.

Utilizando programación orientada a objetos nos permite poder crear una aplicación desde alto nivle, especificando sólo un mínimo de codigo en nuestro objeto programa. Si quisieramos mandar a cualquier dispositivo nuestra imágen, la línea de despliega puede cambiarse a algo más general, aprovechando las propiedades de **herencia** y **polimorfismo** de cada clase de objeto :

```
p : programa (

        i : imagen (    xres=100.
                        yres=100.
                        grises = 256.
                    ).
        d  : camara.
        v  : dispositivo.

        camara.digitaliza(i).
        v = que_dispositivo?
        i.invierte.
        v.despliega(i).
  ).
```

Con esto podemos asegurar que nuestra aplicación sea lo más general posible en cuanto al hardware que se puede utilizar. El elnguaje cuenta con variables especiales para guardar información sobre el ambiente de usuario, cuantos objetos existen, cuantos de algúna clase, etc.

Para generar las tablas del compilador y de codigo final se utilizará una herramienta del sistema operativo UNIX : YACC. Esta herramienta recibe una gramática de entrada y genera una tabla de parse tipo LALR.

El lenguaje fuente de la aplicación final puede ser C++ o C. Para el primero el codigo es inmediato, debido a la orientación de objetos, para el segundo se requiere de una traducción utilizando rutinas de soporte para simular un ambiente de objetos. La aplicación final ejecutable será generada utilizando un compilador para cualquiera de los lenguajes.

II. RUTINAS DE SOPORTE PARA EL S.O.(MEMORIA, E/S).

El ambiente requiere de ciertas rutinas de soporte interno para poder manipular los objetos, el compilador, etc. Estas funciones se dividen en tres grupos :

◊ Manejo de Memoria.
◊ Manejo de I/O (archivos, impresora, etc...).
◊ Interface para llamadas al S.O.

En éstas rutinas se incluyen manejadores de memoria, de disco y llamadas a interrupciones.
El ambiente inicial para todas las partes del proyecto es el sistema operativo MS DOS 3.3, el lenguaje para desarrollo es C, sin embargo se ha comenzado a evaluar otras alternativas sobre el sistema operativo y diferentes implantaciones del lenguaje C.

El problema principal en la manipulación de imágenes son los requerimientos de memoria, para resolverlo se proponen tres soluciones :

◊ **Manejador de Memoria virtual.**
Este permite mover entre memoria y disco los objetos a manipular, permitiendo así de cierta manera la disposición de memoria mayor que la física. El equipo que se utiliza cuenta con discos duros desde 30 hasta 60 megabytes de capacidad, por lo que es bueno considerar la posibilidad de usarlos como extensión de memoria principal.

◊ **Sistemas Operativos Alternos.**
El ambiente de desarrollo actual es DOS 3.3, las limitaciones de memoria y la disponibilidad de maquinas con procesadores más sofisticados y de mayor capacidad (i286,i386) permiten pensar en utilizar un sistema operativo con mayor capacidad. El sistema que se considero para evaluación es MS OS/2 principalmente por su compatibilidad con MS DOS, actualmente se está revisando bibliografía para determinar si su uso puede ser considerado. Otra alternativa la constituyen UNIX y AIX por ser un sistema operativo de gran difusión para estaciones de trabajo, sin embargo se han tenido problemas en su instalación, aunque se toma en consideración para la evaluación.

◊ **Manejador de Memoria utilizando i386.**
En el proyecto se dispone de computadoras IBM modelo PS55sx, éstas cuentan con un procesador i386. Este cuenta con ciertas caracteristicas avanzadas para manejo de memoria y procesos, la posiblidad que existe es el poder desarrollar un manejador que aproveche éstas caracteristicas. El manejador sería un shell pequeño, que permitiera activar procesos, asignar memoria y accesar interrupciones de DOS.

III. FUNCIONES PARA PROCESAMIENTO ESPACIAL

Una de las partes básicas del proyecto lo constituyen en sí una serie de rutinas para procesamiento, juntas forman una biblioteca completa para ser utilizadas en la parte de la aplicación final.

La biblioteca ésta divida en dos tipos de funciones, las primeras son para procesamiento espacial , estas se basan en algoritmos que actúan sobre los puntos originales de la imágen. Los principales algoritmos utilizados en estas funciones son los que engloban las siguientes técnicas:

◊ **Reforzamiento y detección de bordes.**
Esta técnica se basa en la utilización del gradiente y el Laplaciano de una función discreta. Aplicando éstos operadores, las discontinuidades entre los bordes de la imágen se marcan de tal manera que es más sencillo diferenciar diferentes objetos por sus contornos.

◊ **Manipulación de histogramas.**
La distribución de probabilidad de los niveles de gris en una imágen nos dan una forma de medir su constraste. Al manipularlo por medio de herramientas estadisticas es posible hacer más clara un área dada o oscurecer y dar contraste para diferenciar objetos en una imágen.

◊ **Suavizamiento.**
La idea de suavizamiento es el contrario de reforzamiento de bordes, en ésta técnica las discontinuidades son consideradas como ruido en forma de puntos brillantes de mayor magnitud que el promedio de la imágen, para reducirlo se utilizan métodos de promedios y media entre puntos vecinos. La característica principal que tiene este uso es que al reducir las discontinuidades la imágen aparece borrosa, y dependiendo de la consideración de tamaño para puntos vecinos se genera un efecto de cuadricula sobre la imágen.

◊ **Aplicación de mascarillas (esta incluye algunas de las técnicas anteriores).**
Las técnicas anteriores involucran algoritmos especiales para cada una, utilizando el principio del teorema de convolución podemos crear una matriz de n x n para lograr el mismo efecto que la detección de bordes, reforzamiento y suavizamiento.
El problema con el uso de éstas matrices o mascarillas es que en algunos casos el efecto local se marca demasiado en imágenes con resolución baja, por ejemplo para una mascarilla de tamaño 3 x 3 que suavize una imágen por promedios, el efecto de cuadricula (tablero de ajedrez) que se mencionó anteriormente es muy marcado.

◊ **Pseudocolor.**
En algunas aplicaciones se plantea como opción el coloreado de ciertas áreas para resaltar algunos detalles de interes! Esto puede ser hecho de dos formas principalmente :
por nivel, esto es buscar un nivel de gris y cambiar lo que se encuentre en ese nivel a un valor asociado de color, el otro es punto a punto, similar a los paquetes de edición de gráficos comerciales que existen. En éstas rutinas se incluye soporte para manejar toda la capacidad gráfica de los subsistemas VGA y MCGA en el equipo IBM PS.

◊ **Fotomontaje.**

El fotomontaje engloba la edición y restauración de fotografías principalmente. Se desarrollo un algoritmo para recorte de objetos en una imágen, moverlos y posicionarlos en otra diferente.

Este tipo de edición es de utilidad en arquitectura, si se piensa en la posibilidad de unir un sistema de imágenes con un sistema de diseño CAD, para editar modelos reales con esquemas de diseño por computadora.

La biblioteca se encuentra actualmente en formación. Las ampliaciones que se harán serán en cuanto a capacidad para procesamiento y manejo de imágenes más grandes o conjuntos de ellas. Se espera poder utilizar al máximo al capacidad del hardware disponible, tanto de procesamiento como de digitalización y salida.

IV. FUNCIONES PARA PROCESAMIENTO EN FRECUENCIA (FOURIER).

El procesamiento en frecuencia es más complicado que el espacial, debido a que implica una transformación de la imágen como fase previa. La transformación utilizada es por medio del algoritmo de transformada rápida de Fourier con el algoritmo de Cooley Tukey referido en [GONZ87] y [LIM 90]. La codificación de éste algoritmo exige requerimientos de memoria altos, ésta fué la principal razón para considerar otros sistemas operativos y manejadores de memoria.
Una vez procesada la imágen se utilizan filtros para lograr enfatizar ciertas zonas, los filtros ha utilizarse son los siguientes :

- ◊ Pasa Altos.
- ◊ Pasa Bajos.
- ◊ Butterworth.
- ◊ Homomórfico.

Algunos de éstos tienen relación con los procesos espaciales aunque los resultados logrados en frecuencia son mejores. Es posible también crear filtros en frecuencia y realizar una convolución en espacio, es similar a la técnica con mascarillas.

V. FUNCIONES DE SOPORTE PARA VISUALIZACION, IMPRESION, COMPACTACION Y COMUNICACION CON DISPOSITIVOS.

Una vez que se obtienen resultados al procesar una imágen es necesario visualizarlos de alguna forma, se cuenta con dos dispositivos principales de salida para visualización : video e impresora.

En video se incluyen los subsistemas VGA y MCGA para el equipo IBM PS2 y monitores externos con mayor resolución y niveles de gris.

La impresión se realiza por medio de una técnica de sustitución de patrones de puntos por niveles de gris, en éste caso se asigna para cada valor de gris un patron de 8 por 8 pixels a imprimirse.
Los mejores resultados fueron utilizando una impresora Apple Laserwriter II NTX, éste equipo es capáz de imprimir con una resolución de 300 puntos por pulgada.
Se comtempló también la implantación de esquemas de codificación para compactación de imágenes, ya sea para almacenamiento o para transmisión por puertos. Los esquemas utlizados son los siguientes :

- ◊ Codificación Huffman.
- ◊ Codigos B.
- ◊ Codigos de Shift.
- ◊ Codificación Aritmética.

Los primeros códigos son llamados de longitud variable porque se asigna a cada caracter un numero variable de bits dependiendo de su frecuencia de ocurrencia. La última codificación utiliza un enfoque de geometría fractal para compactar una imágen completa en un número de punto flotante.

VI. INTERFASE DEL AMBIENTE FINAL.

El ambiente integrado final, se piensa con una interfase interactiva con ventanas múltiples y ratón como dispositivo de selección. Esta tiene que ser muy amigable para seguir con la filosofía orientada al usuario del proyecto.
La primera versión será orientada a texto, después se pensará en migrar a un ambiente gráfico, la razón es dar más énfasis a otros puntos base del proyecto.

CONCLUSIONES

El uso de técnicas de procesamiento abre una herramienta nueva para diferentes campos que dependen de una imágen de algún tipo como parte importante de los procesos a realizarse, ya sean de diagnóstico como es la medicina o de reconocimiento de formas para la industria.

Esto permite pensar en procesos con una calidad mayor y un control más estricto llevado a cabo con tecnología no importada para America Latina. Esto implica no depender de otros países para utilizar tecnología de vanguardia en procesos industriales y de medicina.

AUTORES y COAUTORES DEL PROYECTO

A continuación se lista a los autores y coautores del proyecto por módulo, todos son estudiantes de 8º y 9º semestres de la carrera de Ingenieria en Sistemas Computacionales en el ITESM Campus Querétaro.

Diseño del Lenguaje y Compilador :
Idalia Constantina Solis Castro.
Francisco Espinola García.

Asesoria en el área de Compiladores y Lenguajes:
Ing. Alfonso Rodriguez.

Manejo de Memoria, Sistema Operativo y Transformada de Fourier :
Ma. Luisa Herrera López.
Concepción Loyola Martínez.
Eva del Carmén de Aquino Rizo.
Alejandro Garza.

Rutinas de Soporte :
Edgar Gerardo Mondrágon Martinez.
Alejandro Fernández Fraga.
Oscar León Espinosa.
Salvador Ugalde Shimano.
Damián Vazquez Soriano.
Gerardo Salazar Rocha.
Mauricio Mejía Alvarez.
Edgar Ortega M.

REFERENCIAS Y BIBLIOGRAFIA

[GONZ87] Digital Image Processing , Gonzalez, C. Rafael. Addison Wesley, 1987.

[BORL88] Turbo C Versión 2.0 Programmer's Guide. Borland International, 1988.

[ELEC89] EDC - 1000 Computer Camera Technical Manual. Electrim Corp., 1989.

[APPL89] Laser Writer II NT NTX Owner's Guide. Apple Computers, 1989.

[DUNC89] Advanced MS DOS, Duncan Ray. Microsoft Press, 1989.

[NORT89] Programmer's guide to the IBM PC & PS/2. Norton, Peter. Microsoft Press, 1989.

[LIM 90] Two Dimensional Signal and Image Processing Lim, S. Jae. Prentice Hall, 1990.

APLICACION DE LA TECNOLOGIA DE SISTEMAS EXPERTOS AL DIAGNOSTICO DE DAÑOS EN ESTRUCTURAS AFECTADAS POR INCENDIOS

Michele Tatoli, Tomás R. Maneiro
Ingeniero Civil, Universidad Central de Venezuela
y
Miguel Cerrolaza
Profesor, Instituto de Materiales y Modelos Estructurales,
Universidad Central de Venezuela, Apto. Postal 50361,
Caracas 1050-A, Venezuela.

RESUMEN

El presente trabajo propone un prototipo de Sistema Experto para la evaluación de Situaciones Críticas en Ingeniería (EECS-1), especificamente se desarrolló una Base de Conocimientos en el área de el Diagnóstico de Daños en Edificaciones de Concreto Armado afectadas por Incendios. El conocimiento fué representado en términos probabilísticos de hipótesis y evidencias, y el motor de inferencia se basa en el teorema de Bayes.

Se desarrollaron tres módulos complementarios del sistema, cuyos objetivos son : justificar las preguntas del EECS-1, ayudar al usuario a la comprensión de las mismas y sugerir correctivos asociados a la situación diagnosticada.

El EECS-1 fué validado con la ayuda del experto humano, mostrando un comportamiento satisfactorio con el juicio de éste, aún en situaciones límites.

DESCRIPCION DEL EXPERTO BAYESIANO.

El Experto Bayesiano es un programa computacional escrito en lenguaje Pascal (Turbo Pascal 4.0) que funciona en entorno de MS-DOS, utilizando el teorema de Bayes como mecanismo de inferencia en procesos de predicción y diagnóstico de situaciones que se presentan en el mundo real. El motor de inferencia trabaja con dos tipos de información, una "a priori" proporcionada por la fuente del conocimiento (expertos humanos, libros, estadísticas, revistas, etc) y otra "a posteriori" aportada por el usuario al sistema durante el diálogo. Adicionalmente el sistema está provisto de tres módulos complementarios : un módulo de Justificación de preguntas que le permite mostrar su línea de razonamiento al usuario cuando éste lo solicite, otro denominado módulo de ayuda, diseñado para explicar o aclarar al usuario las preguntas que hace, y finalmente un módulo de correctivos que orienta al usuario sobre las acciones preliminares asociadas a la situación diagnosticada.

REPRESENTACION DEL CONOCIMIENTO.

El Experto Bayesiano requiere que el conocimiento sea estructurado de una forma particular, por el hecho de basar su mecanismo de inferencia en el teorema de Bayes que maneja información de tipo probabilistico.

En principio, se establecen las hipótesis ó conclusiones que manejará el sistema y sus probabilidades "a priori", que representan las probabilidades de ocurrencia en ausencia de cualquier información adicional. Luego, el experto aporta la probabilidad de que se verifique un hecho ó evidencia supuesto que una hipótesis esté presente $P(E:H)$ ó no esté presente $P(E:\bar{H})$. Estas se denominan probabilidades condicionales y generalmente no son manejadas con facilidad por el experto, ya que su conocimiento suele ser más cualitativo que cuantitativo, sin embargo, con un poco de esfuerzo el experto podrá cuantificar la probabilidad condicional $P(E:H)$. Esto significaría para un médico, por ejemplo, suministrar la probabilidad de que un paciente tenga fiebre(E), si sabe que tiene gripe(H). Lo que realmente le representaría mayor

dificultad sería dar la probabilidad de que el paciente tenga fiebre(E) si éste no tiene gripe(noH), pero afortunadamente existe una expresión matemática que relaciona ambos términos con lo cual se calcula la probabilidad condicional $P(E:\bar{H})$ a partir de la P(E:H) proporcionada por el experto.

En definitiva, la BC queda estructurada en dos partes: la primera contempla una lista de las evidencias que se podrían observar y que llamamos base de evidencias (BE), figura 1 , y la segunda se compone de las hipótesis, sus correspondientes probabilidades iniciales y las probabilidades condicionales (P(E:H) y $P(E:\bar{H})$) de las evidencias asociadas a cada hipótesis, como se muestra en la figura 2 .

Es importante aclarar, que no es necesario que todas las evidencias que constituyen la BE, tengan inherencia en cada hipótesis.

BASE DE EVIDENCIAS (FUEGO).

```
************************************************************

      01  ¿ EXISTEN MANCHAS DE HUMO EN EL CONCRETO ?

      02  ¿ HUBO PERDIDA DEL FRISO EN EL ELEMENTO ?

      03  ¿ SE PRESENTO DESCASCARAMIENTO DEL CONCRETO ?

      .

      .

************************************************************
```

Fig. 1

BASE DE CONOCIMIENTOS (LOSAS)

```
************************************************************

      01    LOSA LEVEMENTE AFECTADA, 0.16 ,

      1 , 0.58 , 0.244 ,

      2 , 0.25 , 0.694 ,

      .

      17 , 0.00 , 0.132 ,

      999

      .

      .

************************************************************
```

Fig. 2

MODULO DE JUSTIFICACION.

La justificación de EECS-1 está basada en la evaluación constante del efecto que una respuesta del usuario causa sobre las probabilidades de las hipótesis ó conclusiones manejadas. A este efecto se le denominó Factor de Importancia (FI). El proceso de diseño de la estructura para el cálculo del FI requiere gran esfuerzo debido a que el mecanismo de inferencia bayesiano, por sus características propias, no se presta para acometer con facilidad esta labor.

Básicamente, el FI es la multiplicación de dos números, donde el primero representa el salto en la probabilidad de las hipótesis, ocasionado por la respuesta del usuario a una evidencia mostrada por el sistema, y el segundo es otro factor al que se denominó Factorfuzzy, ya que proviene de la aplicación de la lógica difusa al sistema.

La aparición de este nuevo factor se debe a otra característica propia del método bayesiano, la cual establece que si una hipótesis tiene una probabilidad de ocurrencia baja, se verá fuertemente influenciada por una

evidencia "a favor" (una evidencia "a favor" es aquella que aumenta la probabilidad de la hipótesis), y por el contrario si la hipótesis presenta una alta probabilidad antes de evaluar el efecto de una respuesta, el desplazamiento que una evidencia "a favor" puede producir es mucho menor. Este fenómeno se agudiza, en la medida que las probabilidades son muy bajas ó muy altas.

Se incorporó al sistema la ecuación de una curva definida a trozos, figuras 3 , 4 y 5 , obtenida empíricamente, que establece los pares ordenados de un conjunto borroso, cuya primera componente es la probabilidad antes de evaluar la respuesta (Pact), y la segunda es el Factorfuzzy. En la curva se puede observar que el Factorfuzzy crece con Pact, con el fin de compensar la situación arriba señalada.

Adicionalmente, se crearon un par de conjuntos de variable lingüística para facilitar la comunicación.

El primero de ellos, figura 6 , establece una variable lingüística a la respuesta numérica del usuario, por ejemplo una respuesta de 3, significa una afirmación con reservas, (La reserva en la afirmación puede provenir de dos hechos, uno, que no se tenga certeza sobre la misma.

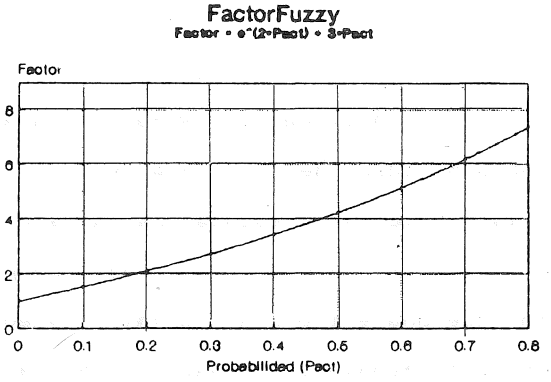


Fig. 3

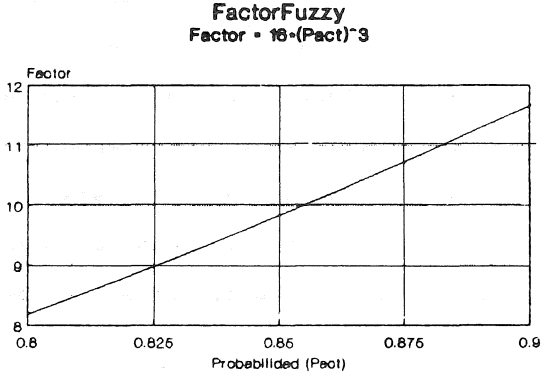


Fig. 4

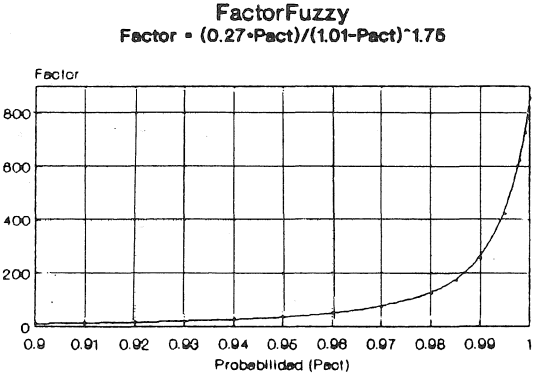


Fig. 5

y dos, que la presencia de la evidencia no es total), de la evidencia que el sistema pregunta.

Respuesta Numérica	Variable Lingüística
5	Afirmación sin dudas
4	Afirmación con fuerza
3	Afirmación con reservas
2	Afirmación débil
1	Afirmación muy vaga
0	Incertidumbre total
-1	Negación muy insegura
-2	Negación débil
-3	Negación moderada
-4	Negación fuerte
-5	Negación rotunda

Fig. 6

El segundo conjunto, figura 7 , se adaptó al FI, estableciendo una relación directa entre éste y una variable lingüística, por ejemplo, a un FI comprendido entre 0.6 y 1, se le asignó la frase " Refuerza en gran medida la importancia de la hipótesis ".

Rango del FI	Variable lingüística
> 1.0	Corrobora en forma tajante
0.6 a 1.0	Refuerza en gran medida
0.4 a 0.6	Aumenta considerablemente
0.2 a 0.4	Acrecienta moderadamente
0.0 a 0.2	Aumenta levemente
0	No influye nada
-0.2 a 0.0	Disminuye levemente
-0.4 a -0.2	Debilita moderadamente
-0.6 a -0.4	Disminuye considerablemente
-1.0 a -0.6	Aminora en gran medida
< -1.0	Reduce en forma tajante

Fig. 7

Es necesario explicar ciertas características que se establecieron para el FI, una de ellas es que puede ser positivo o negativo, dependiendo del sentido en que varía la probabilidad de la hipótesis. Si se incrementa ésta el FI es positivo y si disminuye es negativo. Otra característica es que generalmente oscila entre -1 y 1, lo que no quiere decir que esté acotado entre estos dos valores, por ello existen, en la segunda escala lingüística, dos intervalos que se extienden hasta $\pm\infty$

partiendo de ±1 respectivamente. Los rangos que se establecieron en la escala lingüística, son el resultado de un proceso de ensayo y error con el sistema, hasta lograr adaptar las variables lingüísticas al FI. La estructuración de estos rangos es simétrica, además los intervalos asociados a valores bajos son de menor dimensión comparados a intervalos de valores altos; esto se explica por el hecho de que el FI calculado en un proceso coherente de entrada de información, es decir sin respuestas contradictorias por parte del usuario, tiende a ser bajo y por tanto era preciso afinar la selección de la variable lingüística.

El módulo de justificación presenta un reporte actualizado (las tres últimas evidencias preguntadas por el sistema, las respuestas del usuario a las mismas y el efecto de estas en la hipótesis más probable al momento de solicitar la justificación) que permite explicar al usuario el porqué de su conclusión parcial, lo que hace posible que el usuario se dé cuenta de cuales evidencias son las que orientan al sistema en determinada dirección y la magnitud de esta orientación.

MODULO DE AYUDAS.

Como consecuencia de lo particulares y específicos que suelen ser los problemas que se procesan a través de un SE, surge por parte del sistema la utilización de un vocabulario técnico propio del campo que se está tratando, lo que obstaculiza la comunicación máquina-usuario que en momentos puede dar pie a este último, a mal interpretar el sentido de las preguntas que realiza el sistema. Esto originaria la incorporación de información errónea, provocando cambios inadecuados de la línea de razonamiento y por tanto resultados poco confiables. Es por esto, que surge la necesidad de diseñar la forma de proporcionar al usuario, en cualquier instante que lo requiera, información complementaria de la pregunta que el sistema efectúa.

En principio, la investigación se enfocó a la parte operativa, para crear una infraestructura dentro del programa principal que permitiese acceder a la información complementaria, sin afectar el proceso de inferencia que el sistema desarrolla simultáneamente. En una segunda fase, se recurrió al experto humano, para que éste proporcionara la información adicional en un lenguaje más común, que fuese

capaz de aclarar al usuario cada pregunta; esta información se almacena en un archivo, que contiene los números de las evidencias y sus explicaciones correspondientes, aquellas preguntas muy explícitas no necesìtan ser incluídas.

MODULO DE CORRECTIVOS.

El módulo de correctivos surge debido a una sensación de vacío que se generaba cuando el sistema realizaba su diagnóstico y finalizaba así su labor. En este momento era deseable que adicionalmente al diagnóstico, el sistema sugiriera las acciones a tomar asociadas a éste.

Es evidente que estas recomendaciones solo pueden ser acciones preliminares y generales, sin embargo, permiten orientar al usuario y aumentan su percepción del problema diagnosticado.

El módulo de correctivos es accesible al usuario a través de un menú que el sistema presenta una vez emitido su diagnóstico y para su funcionamiento requiere de una base con información proporcionada por el experto.

DESCRIPCION DEL EECS-1.

La frecuencia con que se presentan incendios en zonas urbanas y la dificultad que significa para la mayoría determinar la capacidad estructural remanente después del hecho, justificaba el desarrollo de SE en el área. Otra razón era que la solución del problema requiere de una buena parte de heurística y manejo de información imprecisa y más importante aún, se disponía de un experto humano en el área dispuesto a colaborar.

El primer paso en la elaboración de la BC, fué la identificación del problema, caracterizando los aspectos más importantes, además de fijar los objetivos y alcances del sistema. Es aquí donde surge el primer obstáculo, ya que el conocimiento no se adaptaba plenamente a la estructura que caracteriza al Experto Bayesiano, lo que obligó a reformular el planteamiento inicial del problema, que consistía en el diagnóstico del daño de la estructura como un todo, cambiando al diagnóstico del daño en los elementos estructurales por separado. Esto condujo a la elaboración de tres BC asociadas a losas, vigas y columnas respectivamente.

Se acordó que las BC debían contemplar como hipótesis los niveles de daño posible. El experto creyó conveniente establecer cuatro hipótesis, que son : Levemente afectada, moderadamente afectada, fuertemente dañada y severamente dañada, con lo que se cubría el espectro de daños totalmente.

Se elaboró una lista tentativa de hechos presentes en una edificación siniestrada, que permiten al experto hacer conjeturas acerca del estado de la misma y que debían ser preguntadas por el sistema al usuario, con el fin de elaborar el diagnóstico, a partir de esta lista se constituyó la base de evidencias definitiva que complementa la BC.

Cada una de las BC se fueron nutriendo lenta y cuidadosamente con el conocimiento del experto, traducido éste en probabilidades que el sistema bayesiano emplea en su mecanismo de inferencia.

EJEMPLO REPRESENTATIVO.

Se presenta a continuación un ejemplo que permite demostrar el comportamiento del sistema, ante situaciones extremas e intermedias. Las preguntas del sistema van precedidas por la identificación "EECS-1 : ", mientras que, las respuestas del usuario se anteceden con la palabra "Usuario : " .

El ejemplo supone la existencia de una viga de concreto armado con signos muy marcados de deterioro, luego de un incendio. A continuación, se reproduce el diálogo máquina-usuario, de este caso :

EECS-1 : ¿ Se observó acero "no estructural" fundido ?
USUARIO : 5

EECS-1 : ¿ Existen fisuras en el elemento ?
USUARIO : 5

EECS-1 : ¿ Extinguieron el fuego los bomberos ?
USUARIO : 5
EECS-1 : ¿ Hubo pérdida del friso en el elemento ?
USUARIO : 5

(Este diálogo continúa, incorporándose al sistema más información "a posteriori" y cuando éste lo estima conveniente, emite su diágnostico.)

.

.

EECS-1 : ¿ Notaste pérdida de adherencia acero-concreto ?

USUARIO : 5

del diálogo ....

Mi DIAGNOSTICO es :

VIGA SEVERAMENTE DAÑADA

Con una probabilidad de 95 %

Presione ENTER para continuar

En el ejemplos anteriores se pone en evidencia que el sistema es apto en el reconocimiento de los límites del dominio del conocimiento. Para confirmar el buen funcionamiento global del EECS-1, es necesario demostrar que éste no oscila alrededor de situaciones intermedias; en función de esto, se realizó un gráfico para cada BC (losas, vigas y columnas), que se compone de la siguiente manera :

En las ordenadas se representan, los valores de las probabilidades finales de cada hipótesis, luego de contestarse con un valor "fijo" a todas las preguntas realizadas por el sistema. Se entiende por valor "fijo", un valor cualitativo constante del estado de daño del elemento. En las abscisas, se representan las respuestas del usuario.

En estos gráficos, figuras 8 , 9 y 10, se puede observar que el EECS-1, tiene un comportamiento sin fluctuaciones y converge asintóticamente hacia los límites esperados. Otra característica apreciable en los gráficos, es su criterio conservador ante las situaciones que se le plantean, sobre todo si ésta es un poco incierta.

Comportamiento del EECS-1
Elemento : LOSA

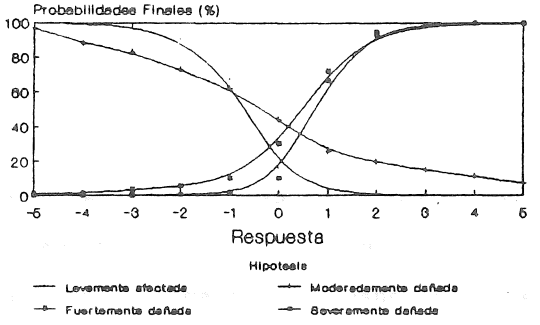


Fig. 8

Comportamiento del EECS-1
Elemento : COLUMNA

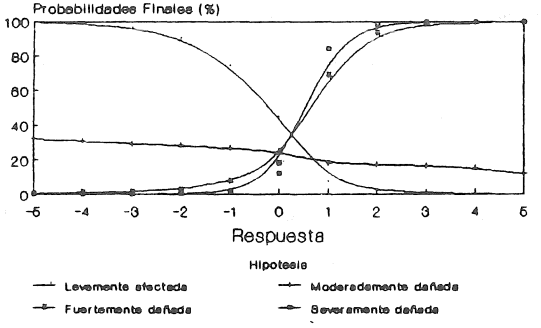


Fig. 9

Comportamiento del EECS-1
Elemento : VIGA

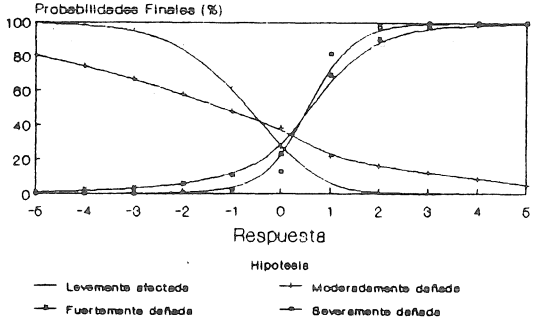


Fig. 10

CONCLUSIONES

La base de conocimientos desarrollada en este trabajo, permite al EECS-1 evaluar los daños en estructuras de concreto armado afectadas por incendio, lo cual tiene una utilidad práctica evidente.

Los módulos complementarios del SE Bayesiano mejoran sustancialmente la comunicación máquina-usuario. El módulo de justificación permite al usuario comprender la línea de razonamiento del sistema, a través de conjuntos de variable lingüística. El módulo de ayudas facilita al usuario la compresión de términos técnicos. El módulo de correctivos complementa el diagnóstico del sistema, con recomendaciones de tipo preliminar asociadas al nivel de daño.

El sistema presentado en este trabajo demostró, en el proceso de validación, tener un comportamiento adecuado y sin saltos, coincidiendo regularmente con el criterio del experto humano.

REFERENCIAS

CERROLAZA, M., GOMEZ, B. y ALARCON, E. (1988)
" Expert System for Quality Prediction in Structural Engineering ". IABSE Congress, Helsinky, Finlandia.

- DELFIN, F., GOMEZ, E. y TORRES, O. (1983)
" Efectos del fuego en estructuras de hormigón "
Revista del IDIEM, Chile.

- HAYES-ROTH, F., WATERMAN, D. y LENAT, D. (1983)
" Building Expert Systems "
Addison Wesley, New York, USA.

- NAYLOR, C. (1983)
" Build your own Expert System "
Sigma Technical Press.

- WATERMAN, D.A. (1986)
" A Guide to Expert Systems "
Addison-Wesley Publishing Company, USA.

Derivación de secuencias de test para protocolos de comunicaciones

Lic. Leonardo Marty †

Ing. Fernando Brum††

Resumen

En los últimos años se ha incrementado abruptamente el uso de redes de datos públicas que permiten interconectar sistemas heterogéneos. Los sistemas conectados a una red se comunican entre sí mediante un conjunto de reglas comunes que se denominan protocolos.

La implementación de un protocolo es una tarea verdaderamente compleja que requiere gran esfuerzo. Dicha implementación se obtiene de una especificación estándar permitiendo derivar varias implementaciones diferentes.

Estas razones motivan la necesidad de servicios de test cuyo objetivo es establecer si una implementación concreta satisface su definición. En este trabajo se intenta automatizar el proceso de derivación de test a partir de la especificación, mediante la construcción de un conjunto de algoritmos que abstraigan el procedimiento ejecutado por un experto humano.

† Escuela Superior Latinoamericana de Informática (ESLAI). Parque Pereyra Iraola, Prov. de Buenos Aires, Argentina.
Ayudante del Instituto de Computación de la Facultad de Ingeniería, Av. Julio Herrera y Reissig 565, Montevideo, Uruguay. (598) (2) 71-42-44/47
Dirección electrónica: marty@incouy.edu.ar

†† Profesor del Instituto de Computación de la Facultad de Ingeniería, Montevideo, Uruguay.
Dirección electrónica: brum@incouy.edu.ar
Director del proyectos de INTERFASE S.A., Zabala 1378, Montevideo, Uruguay. (598) (2) 96-11-43. FAX (598)(2) 96-29-65.

1. Introducción

En los últimos años, se ha incrementado abruptamente el uso de grandes redes de datos públicas. Estas, permiten interconectar sistemas heterogéneos para soportar aplicaciones distribuídas.

Los sistemas adheridos a una red, se comunican entre sí utilizando un conjunto de reglas comunes y convenciones que se denominan protocolos. El modelo de referencia ISO/OSI [ISO 83], define una arquitectura de siete niveles con el fin de estandarizar los sistemas de comunicaciones.

La implementación de un protocolo, es una tarea verdaderamente compleja que requiere, por consiguiente, un gran esfuerzo. Dicha implementación, se obtiene generalmente de una especificación estándar (informal o formal), por lo tanto, estos documentos pueden derivar a varias implementaciones distintas.

Estas razones, sumadas al creciente número de equipos heterogéneos que se pueden interconectar hoy en día, motivan la necesidad de servicios de tests. Una secuencia de tests para un protocolo, tiene por objetivo establecer cuándo una implementación concreta : IUT (Implementation Under Test), satisface su definición. La realización de estos tests implica muchas dificultades, no sólo técnicas, sino tambien políticas y legales. ISO ha intentado establecer un orden en este caos, mediante la publicación de un estándar compuesto de 5 documentos distintos [OSI 88] [OSI 89] [OSI 89a].

La derivación de una secuencia de tests adecuada, a partir de la especificación del protocolo, es un proceso largo y tedioso, realizado generalmente en forma manual por un experto humano.

En este trabajo, se intenta automatizar este proceso, mediante la construcción de un conjunto de algoritmos que abstraigan el procedimiento ejecutado por el experto humano.

El enfoque seguido, intenta mantener una cierta independencia del lenguaje de especificación, aunque se basa firmemente en el modelo de

máquinas de estados extendidas. Para obtener esta independencia, se realiza la derivación a partir de un grafo asociado al protocolo estudiado, y no directamente de la especificación del mismo. La obtención del grafo a partir de la especificación en cualquier lenguaje basado en un enfoque de máquinas de estados extendidas como puede ser SDL o ESTELLE, se puede realizar automáticamente en forma muy sencilla.

2. Sistemas de Tests de Protocolos

Existe una gran variedad de sistemas de test que se pueden aplicar sobre una implementación de un protocolo. En este trabajo se intenta testear protocolos que siguiendo el modelo OSI, se encuentran estructurados en diferentes capas, testeando una sola capa o nivel por vez. Este tipo de tests, se denominan SLCT (Single Layer Conformance Tests).

Existen distintos métodos para la realización de este tipo de tests:

- Método local: En este caso el sistema de test, tiene acceso directo a todas las interfases de la implementación.
- Método distribuido: Cuando se tiene acceso a la interfase superior de la IUT, pero no a las interfases inferiores de la IUT (exceptuando el Nivel Enlace).
- Método remoto: No se posee ningún acceso a las interfases superiores de la IUT, solamente se tiene acceso a la interfase inferior de la IUT.

El primer método es el utilizado por el propio implementador del sistema, quien tiene acceso a todas sus componentes. El segundo método surge, por ejemplo, cuando se contrata a una entidad externa para que realice el test. En ese caso no se le permite al sistema de test, el acceso a toda la implementación, sino a las interfase superior e inferior del mismo.

El método remoto, como su nombre lo indica es el que se utiliza cuando se desea realizar un test a distancia de la IUT. En ese caso no se tiene más acceso

que al nivel físico, a través de un cierto medio de comunicación. Este es el método más utilizado para el testeo de los protocolos de la familia X.25 [Lin 90]. Este será el método de test adoptado en este trabajo.

Los tests que se pueden realizar sobre la IUT se pueden clasificar en tests dinámicos o estáticos. Los aspectos estáticos se concentran en la correcta formación de las PDU (Protocol Data Units) y en la consistencia entre la implementación y las opciones de servicios que dice ofrecer la IUT. Los aspectos dinámicos de conformance se centran en el comportamiento de la IUT al interactuar con otras implementaciones (en particular el sistema de test). Son los tests dinámicos los que se generarán automáticamente.

Existen varios trabajos publicados sobre generación automática de secuencias de test a partir de especificaciones de protocolos con máquinas de estados (FSMs Finite State Machines) [N&M 81] [S&D 85] [Gon 70] [Cho 78]. Un estudio comparativo detallado se encuentra en [S&L 89].

El problema en estos trabajos, no radica en el método seleccionado, sino en el mecanismo de especificación utilizado. Las FSMs no tienen la expresividad suficiente. El problema de la explosión de estados, hace inmanejable aún la especificación de pequeños protocolos. Además, estos mecanismos no poseen forma de expresar eventos dependientes de timers, o canales de comunicación, etc.

Sin embargo los métodos de generación utilizados, ofrecen un conjunto de ideas muy buenas. El enfoque a seguir consistirá entonces en extender estas ideas para aplicarlas a especificaciones en lenguajes como SDL [Bel 88] [CCI 88] o ESTELLE [B&D 88] [ISO 86]. Estos lenguajes ofrecen EFSM (Extended Finite State Machines) como forma de escribir procesos, que en nuestro caso serán las capas del protocolo dado.

Se debe estudiar entonces, en qué forma estos lenguajes extienden las FSMs, y de qué forma estas extensiones afectan los mecanismos de generación de tests propuestos.

3. Características del enfoque presentado

Un experto humano, al generar una secuencia de test, debe ir llevando en su mente, el estado de la IUT, entendiendo por estado, el valor que tienen determinadas variables, o el rango de valores en el que puede estar el valor de una variable en un momento dado.

Esta necesidad de tener en cuenta de alguna forma el valor de las variables y la forma en que son alteradas, es la causa de la introducción de un análisis semántico de la especificación.

Se entiende por análisis semántico de la especificación, a un análisis que requiere conocimientos sobre los dominios de valores a los que pertenecen las variables, y sus propiedades. Existen varios trabajos en este tema que realizan un análisis puramente sintáctico [Ura 87][P&G 90][Arm 89].

El proceso a automatizar es sumamente complejo, por lo que se hace muy difícil una automatización total del mismo. La idea que se intenta seguir, consiste entonces en intentar obtener por ejemplo, un noventa por ciento de las secuencias que un experto generaría manualmente, en forma automática.

Se darán entonces un conjunto de reglas prácticas y heurísticas, que aplicadas en los algoritmos de generación, realizarán búsquedas inteligentes de las secuencias de tests deseadas, imitando la forma de pensar de los expertos, y no realizando búsquedas exhaustivas que pueden llevar demasiado tiempo o no acabar.

De esta manera, los algoritmos presentados siempre terminan retornando uno de estos tres diagnósticos:

- EXITO: Si se completó satisfactoriamente la búsqueda solicitada.
- FRACASO: Si el algoritmo falla, no encuentra la secuencia de test buscada, pero no puede afirmar que dicha secuencia no existe.
- IMPOSIBLE: Si demuestra que la secuencia solicitada, no existe.

4. El Modelo Utilizado

Para mantener una cierta independencia del lenguaje de especificación utilizado, se presenta un modelo que abstrae, de la especificación de un

protocolo la información relevante para la generación de secuencias de tests.

Esta abstracción consistirá en manipular un protocolo como un grafo dirigido. La obtención del grafo asociado a una especificación se puede hacer en forma mecánica de una manera muy sencilla.

DEFINICION 1:

Una *especificación* E de un protocolo es una 4-upla compuesta por:
$E = < S, C, V, T >$ donde:

- $S(E) = S$ es un conjunto de estados.
- $C(E) = C$ es un conjunto de canales tipados.
- $V(E) = V$ es un conjunto de variables tipadas.
- $T(E) = T$ es un conjunto de transiciones.

El conjunto de estados, representa a los estados sintácticos ya sea de una especificación SDL o ESTELLE.

Los canales de comunicación representan o bien las *signal routes* asociadas a un proceso SDL, o los *ips* asociados a un modulo ESTELLE.

Tanto los tipos de las variables, como de los canales asociados a una especificación, se hacen accesibles a través de funciones que se llamarán: *TipoCan* y *TipoVar*.

Las transiciones de una especificación, representan la abstracción de una transición sintáctica de SDL o ESTELLE.

DEFINICION 2:

Una *transición* t es una 5-tupla formada por:

- FROM $(t) = s_0 \in S(E)$.
- TO $(t) = s_1 \in S(E)$.
- WHEN $(t) = < c, s_1(x_1,...,x_n) > \cup \{ \varepsilon \}$
  con $s_1(x_1,...,x_n)$: $TipoCan(c)$ / $c \in C(E)$
- PROVIDED $(t) = pred(x_1,...,x_n)$ con *pred* un predicado
- BEGIN-END $(t) =$ *accion*

Las dos primeras componentes de una transición corresponden a estados sintácticos de la especificación ya sea SDL o ESTELLE. La primera indica el estado origen de la transición y la segunda el estado final de la misma.

La componente WHEN indica la señal que activa la transición. El caso en

que WHEN (t) = ε corresponde a una transición espontánea [Ora 88].

La componente PROVIDED indica la condición que se debe verificar para que una transición esté habilitada para ser disparada.

La última componente corresponde a la acción a realizar en caso de seleccionar dicha transición. Una acción consta de dos partes:

- Un conjunto de asignaciones.
- Un conjunto de mensajes a enviar (OUTPUTs).

Se referirá a estos por las funciones: Asignaciones (t) y OUTPUTs (t) respectivamente.

La nomenclatura utilizada deriva de ESTELLE. Del mismo modo se podría haber tomado de SDL (STATE, NEXTSTATE, INPUT, PROVIDED, TASK).

Observar que se está pidiendo, por las restricciones impuestas, que esta especificación se encuentre en forma normal. Propuestas de cómo llevar una especificación a su forma normal se presentan en [Sar 85].

A partir de una especificación E, se puede obtener un grafo asociado del mismo modo que para las FSMs.

DEFINICION 3:

Se define el *grafo* asociado a una especificación E como
$G = ((E), T(E), \varphi)$ donde : φ: $T(E) \rightarrow S(E)^2$ /
$\varphi (t) =$ FROM (t) x TO (t).

Este grafo, no se debe confundir con la FSM subyacente, sino que, en cierto sentido es más potente, por las extensiones introducidas.

En un primer lugar, este grafo se diferencia de una máquina de Mealy, en que tiene asociado la noción de canal de comunicación. Una máquina de estados extendida, se puede comunicar con más de una entidad al mismo tiempo, por lo que cada mensaje debe ser cuantificado con un canal de comunicación como forma de indicar a quien se dirige y de quien proviene.

Se introduce también la noción de Timer, o de disparo de transiciones por tiempo. Se puede ver a las operaciones sobre Timers como una forma más

de envío y recepción de mensajes, definiendo al Timer como un canal de comunicación. Las operaciones *set* y *reset* equivalen a enviar mensajes a un cierto proceso por el canal TIMER, el cual responde por el mismo canal con mensajes llamados *time-outs* .

Esto implica que en la componente WHEN de una transición puede tener como canal asociado a: TIMER, en este caso, el mensaje corresponde a un *time-out* .

Esta noción puede verse también como una forma de introducir no-determinismo [Ora 88], en el sentido que estas transiciones pueden ser disparadas, fuera del control del sistema de test.

La mayor diferencia con las máquinas de Mealy radica en la introducción de variables. La idea básica consiste en que las variables de condición se introducen como forma de evitar el problema de explosión de estados. La técnica adecuada será entonces, introducir las variables en la menor dosis posible, sólo en los lugares que no utilizarlas provocaría una explosión en la cantidad de estados del sistema.

De este modo un estado de la especificación puede representar muchos estados de la FSM subyacente. La componente PROVIDED de una transición, simboliza la selección entre esa cantidad de estados de la FSM condensados en un único estado de la especificación. Del mismo modo, la acción a ejecutar dentro de la transición, al alterar las variables, representa parte de la noción de cambio de estado en la FSM subyacente.

De ninguna manera el enfoque a seguir será expandir este grafo a la FSM subyacente ya que ésto implicaría caer en el problema que se intentó solucionar al extender las FSM, la explosión de estados. La idea será más bien, realizar búsquedas inteligentes de caminos, y no búsquedas exhaustivas.

Otra forma en la cual se extendieron las FSMs fue cambiando la noción de mensaje. Estos pasaron, de ser manipulados como elementos de un conjunto finito, a ser manipulados mediante la noción de tipo. Esto permite por ejemplo, poder utilizar *pattern-matching* sobre los mensajes para distinguir un

subconjunto específico de mensajes, etc.

A continuación se definen algunos conceptos necesarios para el diseño del conjunto de algoritmos.

Se notará como *Val* al conjunto de todos los valores posibles que pueden tomar las variables de la especificación. Más adelante se definirá este conjunto con más detalle.

DEFINICION 4:

Se define una *asignación* como una función :
σ: V (E) $\rightarrow$ *Val* $\cup$ { $\uparrow$ } / σ (x) $\in$ *TipoVar* (x).

Las asignaciones de variables a valores se extienden de la forma usual sobre los términos utilizados en las diferentes componentes de las transiciones.

La restricción de funciones sobre subconjunto $V(E) \supset X$ se notarán como $\sigma \mid X$.

Se introduce entonces, la noción de estado de la IUT como :

DEFINICION 5:

Un *estado* en el grafo asociado es un par $< s , \sigma >$ donde
$s \in S(E)$ y σ una asignación.

No se debe confundir la noción de estado de la IUT, con la noción de estado de la especificación notado como s. Se llamará al primero estado semántico y al segundo estado sintáctico de la especificación.

Esta noción de estado, refleja la idea de estado de la FSM subyacente al grafo dirigido, donde un estado se debe dar no sólo por el estado sintáctico de la EFSM, sino también por el valor que tiene cada una de las variables.

DEFINICION 6:

Se define el conjunto de las *Variables Alteradas* por una transición como : VarsAlts (t) = { $x \in$ V (E) / x <-
exp $\in$ BEGIN-END (t) }

DEFINICION 7:

Se define el conjunto de las *Variables Locales* a una transición como:
VarsLocales (t) = Variables (WHEN (t))

DEFINICION 8:

Se define el conjunto de las *variables de condición* de una transición como: VarsCondición (t) = Variables (PROVIDED (t))

DEFINICION 9:

Se dice que un estado $< s , \sigma >$ *satisface* una transición t, lo que se anota como:

$< s , \sigma > \models t \leftrightarrow$
FROM (t) = s &
$\exists \sigma_1 / \sigma((\sigma_1 |$ VarLocales (t)) (PROVIDED (t)) = True.

Se dice que la asignación σ_1 satisface t en el estado $< s , \sigma >$. Esto se notará usualmente de la siguiente manera: σ_1: $\pi \models t$. Que un estado satisfaga una transición, significa que si la IUT se encuentra en dicho estado, entonces la transición está habilitada para ser disparada. La señal que se debe enviar para provocar esta transición debe ser: σ (WHEN (t)), con σ una asignación que satisfaga a t.

Los caminos en el grafo $G(E)$ se notarán como $(t_1,...,t_n)$. Esta secuencias de transiciones debe cumplir que:

$$\forall i \; 1 <= i <= n \quad \text{FROM } (t_i) = \text{TO } (t_{i-1})$$

DEFINICION 10:

Se define la *función de transformación* τ como la función que dado un estado, una transición y una asignación que satisface la transición en ese estado, retorna el nuevo estado al que se pasa luego de ejecutar las acciones de la transición escogida.

La definición de esta función, depende de las acciones que se permitan ejecutar en cada transición. Esto puede abarcar desde asignaciones a estructuras de control más complejas.

DEFINICION 11:

Se dice que una secuencia $(t_1,...,t_n)$ de transiciones es un *camino efectivo*

$$\text{desde } \pi_0 = \langle s, \sigma \rangle \text{ a } t \leftrightarrow$$
$$\exists (\pi_0, \pi_1, \ldots, \pi_n) \text{ y } (\sigma_1, \ldots, \sigma_n) /$$
$$\sigma_{i-1}, \pi_i \models t_i \;\&$$
$$\pi_n \models t \;\&$$
$$\pi_i = \tau(\pi_{i-1}, t_i, \sigma_i)$$

La idea consiste en que un camino efectivo es un camino en la FSM subyacente, son los caminos que efectivamente se pueden recorrer en el grafo dirigido asociado.

5. Los Algoritmos de Generación

5.1. Introducción

En esta sección se presentan un conjunto de algoritmos para la generación automática de las secuencias de tests, a partir del grafo asociado a una especificación.

Las secuencias que se intentan generar, son los tests de comportamiento, que se encargan de verificar que cada transición esté correctamente implementada. Es decir que para cada transición de la especificación se generará un test, que consiste en provocar en la IUT la ejecución de la misma y estudiar su comportamiento.

Aplicando las ideas propuestas para FSMs, para cada transición t se generará una secuencia de la siguiente forma:

$$\bullet \text{ PREAMBULO} \propto \text{TEST} \propto \text{CHEQUEOFINAL}$$

donde $\propto$ denota la concatenación de secuencias.

A pesar de que se hable continuamente de secuencias de test, el término correcto debiera ser árboles o digrafos. De todos modos el término secuencia hace alusión a que sólo un camino, el que conduce al test, es el que verdaderamente interesa de la estructura.

DEFINICION 12:

Un *preámbulo* consiste en un camino efectivo en el grafo subyacente a la especificación, desde un estado inicial, a la transición a testear.

El test consiste simplemente en enviar el mensaje: σ(WHEN(t)) donde

σes una asignación que satisface *t*.

El mayor problema a resolver, en la ejecución de esta parte del test, es resolver que diagnóstico emitir una vez que se obtiene una respuesta de la IUT al estímulo enviado.

El chequeo final consiste en verificar que la ejecución de la transición *t* dejó a la IUT en el estado TO (*t*).

La generación de este tipo de tests, implica resolver los siguientes problemas :

- Encontrar el preámbulo adecuado. Como se necesita que se satisfaga la condición del PROVIDED de la transición a testear, dependiente de las variables, ya no es suficiente un análisis puramente sintáctico, sino que también es necesario un análisis semántico. Se necesita manejar la noción de estado definida en la sección anterior.
- Emitir un diagnóstico. Una vez que se lleva a la IUT al estado deseado, es necesario analizar todas las respuestas posibles debido al no-determinismo inherente a la especificación para emitir un correcto diagnóstico acorde a la respuesta obtenida.
- Distinguir cuando una transición es testeable o no. Gran cantidad de transiciones pueden depender de eventos no controlables por la IUT por lo que no es posible testearlas. Incluso estados enteros pueden ser inaccesibles, para el sistema de test.
- Seleccionar un lenguaje adecuado para la descripción de las secuencias generadas.

5.2. TTCN

Con el objetivo de tener un lenguaje estándar para la especificación del comportamiento de los sistemas de test, y las entidades a ser testeadas, la ISO definió una notación llamada TTCN (Tree and Tabular Combined Notation) estandarizada en [OSI 89].

Esta notación aspira a ser lo suficientemente general como para permitir describir tests para cualquier protocolo OSI. TTCN no tiene la intención de ser un lenguaje ejecutable, sino que esta notación permite definir una secuencia de test abstracta, independiente de la implementación. Cada secuencia abstracta se compone de una sucesión de eventos atómicos con los suficientes

detalles como para juzgar el comportamiento de una implementación del protocolo y formular un diagnóstico (PASS, FAIL, INCONCLUSIVE).

5.3. Supuestos básicos

El conjunto de algoritmos presentados, se basan en ciertos supuestos básicos. La intención es que estos supuestos sean lo suficientemente amplios como para no excluir ningún protocolo OSI.
Estos supuestos son los siguientes:

- Los tests a realizar serán tests remotos en los que el sistema de test, solamente tiene acceso a la interfase inferior de la IUT. Por esta razón, se debe indicar en la información de la arquitectura del test el nombre del canal de comunicación, controlable por el sistema de test. A este canal se le denominará : c
- La probabilidad de ruido en la línea de test es despreciable. Este es el supuesto más fuerte que se realiza, porque no siempre es válido, pero esta suposición simplifica enormemente el trabajo.
- Existe una secuencia que se llamará *ri* (*Reset Input*) y un estado $\pi_{inic} = < s_{inic}, \sigma_{inic} >$, tal que dicha secuencia aplicada en cualquier momento, conduce a la IUT al estado inicial π_{inic}.
- El predicado $\models$ es decidible, existiendo un algoritmo para encontrar en cada caso, una asignación que satisface la transición para el estado dado.

Este último supuesto se introduce por utilizar un análisis semántico sobre la especificación del protocolo.

El pedir decidibilidad para el predicado $\models$ puede parecer restrictivo, pero la experiencia demuestra que no lo es. Incluso, en la práctica, se trabajó con restricciones mucho más fuertes que esta, como exigir que las variables de condición sean de algún tipo isomorfo a Z_n con $n \in N$. Esto permite restringir los predicados a utilizar a los operadores de comparación respectivos y las expresiones en las asignaciones, a sumas y productos. Esta condición más restrictiva que la original, sobra para todos los protocolos experimentados.

Se explica fácilmente al observar la naturaleza de las variables utilizadas en la componente PROVIDED de las transiciones, porque alcanza con

pedir dominios isomorfos a Z_n.

Estas variables son generalmente:

- booleanas: isomorfas a Z_2. (Ej : I'm Busy, etc)
- contadores de reintentos: isomorfo a $Z_{Cantintents}$.
- variables de numeración de PDU : como ser vs, vr.

Esta restricción transforma al conjunto *V*, en el conjunto de los naturales. De todos modos esta restricción no impide la utilización de tipos más complejos como *buffers* o *stacks*, puesto que generalmente la condición que se testea sobre estas variables son por ejemplo sobre su largo, (si está lleno, si está vacio), condición que se puede representar con una variable del tipo deseado.

5.4. Detección de Estados y Transiciones Inaccesibles

Existen muchas transiciones inaccesibles debido a eventos no controlables en la IUT. A grandes rasgos, se pueden distinguir:

- Variables no controlables. Las condiciones dependientes de la implementación como ser que una IUT se encuentre *busy*, se expresaron en la especificación como variables a las que se consulta su valor, pero nunca se modifican. Esas variables, se pueden detectar con un análisis sintáctico de la especificación, siendo las transiciones que dependen de estas variables en sus condiciones inaccesibles.
- Canales no controlables. Por asumir que el test es remoto controlando únicamente el canal c_r, no es posible provocar las transiciones que dependen de recepción de mensajes por canales diferentes a c_r. Estas transiciones tampoco serán controlables.
- Estados inaccesibles. Existen estados que no son accesibles porque todas las transiciones que conducen a él son inaccesibles. A su vez todas las transiciones originadas en esos estados, serán inaccesibles.

Estas condiciones, se combinan entre sí, generando más transiciones inaccesibles. Se presenta a continuación un algoritmo para detectar transiciones no testeables.

La idea para detectar todas las transiciones inaccesibles para el sistema de test, consiste en marcar las transiciones que son directamente inalcanzables (las que dependen de canales o variables no controlables, etc.), lo que

permite visualizar qué estados son inaccesibles. De esta manera se van derivando nuevas transiciones inalcanzables (las que tienen su origen en estados ya marcados) y a su vez nuevos estados inaccesibles. Al finalizar este proceso se tienen marcadas un conjunto de transiciones que son inaccesibles al sistema de test.

Una transición marcada por este algoritmo es necesariamente no testeable, pero ésta no es una condición suficiente. Por ejemplo, en el caso de testear el nivel 2 del protocolo X.25, no se puede obligar a la IUT a emitir tramas de información, por lo tanto no se puede alterar la variable que cuenta las tramas emitidas en la IUT. Este caso no es detectado por este algoritmo.

Un caso particular ocurre con las transiciones que dependen del canal TIMER, o sea las transiciones que dependen de eventos provocados por time-outs. Estas transiciones se consideraron como accesibles, puesto que para provocarlas, simplemente se debe hacer que el sistema de test permanezca sin transmitir PDUs por un tiempo 2d + t, siendo d la demora de la transmisión y t el tiempo de time-out para dicha transición.

A pesar de esto, estas transiciones serán consideradas como incontrolables en el momento de emitir diagnósticos en un test, puesto que pueden ser provocadas al expirar el tiempo de espera, sin que el sistema de test lo desee, debido a demoras en la transmisión de un mensaje.

5.5. Algoritmos para Encontrar Preámbulos

El problema a resolver consiste en encontra un camino efectivo desde el estado inicial π_{inic} a la transición a testear t. Esto implica no sólo encontrar un camino efectivo desde π_{inic} a FROM (t), sino que debe suceder que el estado final de ese camino debe satisfacer la transición t.

Este problema se subdivide a su vez en dos problemas más simples, primero encontrar un camino efectivo desde π_{inic} al resto de los estados de la especificación. Una vez resuelto esto, se desea verificar que la IUT quede en condiciones de poder ejecutar la transición deseada, o sea que se satisfacen la

condición del WHEN.

La búsqueda del camino desde π_{inic} a cada estado de $S(E\ )$, se realiza solo una vez para cada estado, independiente de la cantidad de transiciones que se originan para cada estado. Luego, se debe verificar una vez para cada transición, que la IUT quede en condición de ejecutar la transición a testear. En caso de que no se esté en condiciones, se debe buscar un ciclo en el grafo que permita hacer válida la condición del WHEN de la transición que se desea seleccionar.

Para encontrar un camino desde el estado inicial a cualquier otro estado de $S(E\ )$, se propone aplicar el algoritmo de Dijkstra para encontrar caminos mínimos en un grafo [A&H 76] modificado de manera que los caminos que obtengan sean caminos efectivos.

ALGORITMO 1:

$EstsRevis : P(S(E))$
$Expandir : STACK\ (<S(E\ ), V(E) \rightarrow Val>)$

$EstsRevis := \{ s_{inic} \}$
$Expandir := \{ \pi_{inic} \}$
MIENTRAS (EXPANDIR $\neq \emptyset$) *HACER*
  $< s , \sigma > := Pop\ (Expandir)$
  $\forall t \in T(E) / (< s , \sigma > \models t)\ \&\ not(T0\ (t) \in EstsRevis)$
    $EstsRevis := EstsRevis \cup \{ To\ (t) \}$
    $\sigma_1 := AsignacionSatisfact\ (t, \sigma)$
    $\pi := \tau(< s , \sigma >, t, \sigma_1)$
    $Push\ (Expandir, \pi)$
RETURN (EstsRevis)

Este algoritmo encuentra un camino efectivo desde π_{inic} a cada uno de los estados en el conjunto EstsRevis en el momento de terminar su ejecución.

Este algoritmo utiliza el último supuesto realizado, la decidibilidad del predicado $\models$. Este utiliza también el algoritmo para encontrar la asignación adecuada AsignaciónSatisfact.

Trabajando con la restricción : $\forall x \in V(E\ )\ TipoVar(x) = Z_n (n \in N)$, se pueden dar reglas para decidir que asignación elegir acorde a los intereses del

test. Por ejemplo una buena regla puede ser, probar los casos de borde, que son los casos con más probabilidad de error en una implementación.

Este algoritmo, es una generalización del enfoque puramente sintáctico. La inclusión del análisis semántico en este algoritmo se realiza a través del predicado $|=$. La no utilización de este predicado, transformaría el algoritmo propuesto en el de Dijkstra. No incluir este predicado en el algoritmo 1, sería absurdo porque los caminos generados, no tienen por que ser efectivos, por lo que hay que revisarlos manualmente uno a uno, lo cual se contradice con el objetivo perseguido.

Esto no implica que no se pueda eliminar el análisis semántico, sino que se puede sustituir el predicado $|=$, por alguna condición sintáctica que implique al predicado $|=$. Una condición que se puede usar en este sentido, puede ser: VarLocales(t) $\supseteq$ VarProvided(t).

Utilizando en lugar del predicado $|=$, una condición de este estilo, obliga la obtención de caminos efectivos, lo que es el objetivo perseguido.

Para cada transición a testear, se debe verificar ahora, que el camino que lleva la IUT al estado origen de la transición encontrado por el algoritmo anterior, la deja en un estado π tal que $\pi |= t$.

En caso negativo debe intentar resolver el problema mediante la localización de un ciclo que cambie el estado de la IUT, hasta satisfacer la transición. La búsqueda de este ciclo se realiza mediante la utilización de una heurística adecuada.

Una heurística sumamente elemental, pero que en la práctica resulta realmente efectiva, consiste en localizar un lazo en ese estado donde se alteran las variables de condición de dicha transición.

5.6. Ejecución del Test y Emisión de Diagnóstico

En este punto, se tienen ya las secuencias que llevan a la IUT a un estado π tal que satisface la transición t, que se desea testear. Ahora se debe efectuar el test y emitir un diagnóstico adecuado.

El primer problema a resolver consiste en evitar repeticiones de pruebas sin sentido. Este caso presentado en [P&G 90] y [P&G 90a] [P&G 90b] se da cuando se tienen transiciones idénticas salvo por las acciones no observables por el sistema de test. En este caso se debe realizar una única prueba para todos estos casos.

Para resolver este problema se introduce una noción de orden entre transiciones.

DEFINICION 13:

Se dice que $t <= t_1$ si las transiciones son idénticas, salvo que los OUTPUTs de t que no utilizan el canal de test c_i están incluídos en los de t_1.

La idea es entonces testear solamente las transiciones minimales con respecto a esta relación de orden. Esto evita hacer pruebas semejantes desde el punto de vista del sistema de test, eligiendo las menores como forma de evitar la mayor cantidad de efectos no observables posibles.

De esta manera, para las transiciones seleccionadas, la realización de la prueba consiste simplemente en obtener una asignación σ que satisface t en el estado π y enviar la señal σ(WHEN (t)). De acuerdo a la señal recibida como respuesta, se debe emitir un diagnóstico (PASS, INCONCLUSIVE o FAIL).

Entonces, estando en el estado $\pi = < s , \sigma >$, al ejecutar el test σ_1 (WHEN (t)) para la transición t, el diagnóstico correspondiente se calcula de la siguiente manera:

- Si
  $< c_i, s_1(x_1, ..., x_n) > \in$ OUTPUT (t) &
  $\tau(\pi, t, \sigma_1) = < s_2 , \sigma_2 >$,
  $\sigma(\sigma_1 |$VarLocales (t)) PROVIDED (t)) = TRUE &
  entonces $\sigma_2(s_1(x_1, ..., x_n))$ genera un diagnóstico PASS.
- Si $\exists t_1 \in T(E)$ /
  FROM (t_1) = s &
  WHEN (t_1) = ε &
  $< c_i, s_1(x_1, ..., x_n) > \in$ OUTPUT (t_1) &
  $\sigma(\sigma_1 |$VarLocales (t_1)) PROVIDED (t_1)) = TRUE &
  $\tau(\pi, t_1, \sigma_1) = < s_2 , \sigma_2 >$,
  entonces $\sigma_2(s_1(x_1, ..., x_n))$ provocará un diagnóstico INCONCLUSIVE.

- Si $\exists t_1 \in T(E\ )$ /
  FROM $(t_1) = s$ &
  WHEN $(t_1) = \langle c, s_1(x_1, \ldots, x_n) \rangle\ c \neq c_1$ &
  $\langle c_1, s_2(x_1, \ldots, x_n) \rangle \in$ OUTPUT (t_1)) &
  $\sigma(\sigma_1 |$VarLocales $(t\ ))$ PROVIDED $(t_1)) =$ TRUE &
  $\tau(\pi\ t_1, \sigma_1) = \langle s_3, \ldots, \sigma_3 \rangle$,
entonces $\sigma_3(s_2(x_1, \ldots, x_n))$ genera un diagnóstico INCONCLUSIVE.
- Si $\exists t_1 \in T(E\ )$ /
  FROM $(t_1) = s$ &
  WHEN $(t_1) = \langle c, s_1(x_1, \ldots, x_n) \rangle\ c \neq c_1$ &
  $\langle c_1, s_2(x_1, \ldots, x_n) \rangle \in$ OUTPUT (t_1)) &
  PROVIDED $(t_1) \cap$ VarIncont $(t_1) \neq \emptyset$
  $\tau(\pi\ t_1, \sigma_1) = \langle s_3, \ldots, \sigma_3 \rangle$,
entonces $\sigma_3(s_2(x_1, \ldots, x_n))$ genera un diagnóstico INCONCLUSIVE.
- Si $\exists t_1 \in T(E\ )$ /
  FROM $(t_1) = s$ &
  WHEN $(t_1) = \langle c, s_1(x_1, \ldots, x_n) \rangle\ c \neq c_1$ &
  $\langle c_1, s_2(x_1, \ldots, x_n) \rangle \in$ OUTPUT (t_1)) &
  $\sigma(\sigma_1 |$VarLocales $(t_1))$ PROVIDED $(t_1)) =$ TRUE &
  $\tau(\pi\ t_1, \sigma_1) = \langle s_3, \ldots, \sigma_3 \rangle$,
entonces $\sigma_3(s_2(x_1, \ldots, x_n))$ genera un diagnóstico PASS
- En el resto de los casos el diagnóstico debe ser FAIL.

En esta descripción, el primer item corresponde al caso esperado, el segundo a transiciones espontáneas, el siguiente a transiciones que dependen de canales o variables no controlables y el último al no-determinismo interno explícito de la especificación.

En los casos de no-determinismo explícito de la especificación, no se puede obligar a la IUT a seleccionar una de las opciones dadas. Esto implica que el diágnostico sea PASS, porque hubo un comportamiento correcto, pero eso no implica que las otras opciones no seleccionadas, se encuentren correctamente implementadas.

6. Conclusiones y Extensiones

Estas ideas fueron aplicadas a la generación de las secuencias de test para el protocolo LAPB (nivel 2 de X.25), [CCI 88a] y se compararon con un conjunto de tests generados por un experto humano.

El primer algoritmo, detectó caminos efectivos a 5 de los 9 estados de la especificación y probó que no existían tales caminos para 3 de los estados.

Emitió un diagnóstico de FRACASO solo en un caso.

El segundo algoritmo, encontró el camino efectivo a 83 transiciones en un total de 119, para 35 probó que era imposible realizar un test, fallando nuevamente en un solo caso.

Se generaron entonces 83 secuencias de test para dicho protocolo. Se hace necesario un mecanismo para generar únicamente un subconjunto de las pruebas. Esto se hace evidente en los casos de excepción donde se genera una prueba para cada estado del sistema.

La preocupación debe ser -entonces- limitar la cantidad de pruebas generadas más que generar las pruebas que faltan.

Otra extensión que se debe realizar, consiste en modificar el modelo utilizado para que una asignación permita manejar el caso en el que se conoce el rango de variación de una variable pero no se sabe cual es su valor efectivo. Las asignaciones deben pasar a ser entonces funciones de variables en $P(Val)$.

Previo a implementar estos algoritmos, se realizó una implementación de un sistema de test para las secuencias generadas -en particular- para el LAPB. Dicho sistema se implementó en C++ sobre un ambiente UNIX, diseñado como un servicio para el conmutado URUPAC de ANTEL - URUGUAY.

7. Agradecimientos

A Ana María por su permanente apoyo.

A Ingrid, Jaime y Marcelo por su invalorable colaboración en la corrección y edición del presente trabajo.

8. Bibliografía

[A&H 76] A.V.Aho, J.E.Hopcroft, J.D.Ullman. The Design and Analysis of Computer Algorithms. Addison Wesley Publishing Company, 1976.

[Arm 89] F.Armengaud. Generation de specifications de tests. Validation de logiciels de systemes de telecomunications. Note Technique NT/PAA/CLC/LSC/2234 TELECOM, 1989.

[B&D 88] Budkowski. P.Dembinski. An introduccion to Estelle, Specification Laguaje for distributed systems. Computer Networks & ISDN Systems, vol 14, 1988.

[Bel 88] Ferene Belina. Introduccion to SDL, 1988.

[CCI 88] CITT Recomendation Z.100, Specification and Description Languaje SDL., ITU, X, BLUE BOOK, Annexes A-F to Z.100. Geneva, 1988.

[CCI 88a] CCITT Recomendation X.25, Intefase between Data Terminal Equipment (DTE) and Data Circuit Equipment (DCE) for Terminal Operating in the Packet Mode on Public Data Networks.ITU VIII.BLUE BOOK. Geneva, 1988.

[Cho 78] T.Chow. Testing Software Design Modeled by Finite State-Machines. IEEE Transactions on Software Eng, Vol SE-4, 1978.

[Gon 70] G.Gonene. A Method for the Design of Fault Detection Experiments. IEEE Fault Tolerant Comput, vol C-19, 1970.

[ISO 83] ISO 7498: Open System Inteconnection, Basic Reference Model, 1983.

[ISO 86] ISO. DP 9074 rev. - ESTELLE, A Formal Description Technique based on an Extended State Transition Model, 1986.

[Lin 90] R.J.Linn. Conformance Testing for OSI Protocols. Computer Networks and ISDN Systems, vol 18, pag 203-219. Elsevier Science Publishers, 1990.

[N&T 81] S.Naito. M.Tsunoyama. Fault Detection for Sequential Machines by Transmition Tours. Proc. IEEE Fault Tolerant Computers, Conf, 1981.

[Ora 88] F.Orava. Formal Semantics of SDL Specification Proc. on Protocol Specification, Testing and Verification VII. pag 143-158, 1988.

[OSI 88] Information Processing Systems - OSI Conformance Testing, Methodology and Framework. ISO/IEC JCT 1/SC 21 DIS 9641. Partes 1-2, 1988.

[OSI 89] Information Processing Systems - OSI Conformance Testing, Methodology and Framework. ISO/IEC JCT 1/SC 21N3077. Parte 3, 1989.

[OSI 89a] Information Processing Systems - OSI Conformance Testing, Methodology and Framework. ISO/IEC JCT 1/SC 21DIS 9646. Partes 4-5, 1989.

[P&G 90] Marc Phalippou. Roland Groz. From Estelle Specifications to Industrial Test Suites, Using an Empirical Approach. Forte 90 1990

[P&G 90a] Marc Phalippou. Generation de Suites de Tests TTCN pour le LapD a partir d'Estelle. Journees Firtech, 1990.

[P&G 90b] Marc Phalippou, Roland Groz Evaluation of an empirical approach for computer-aided test case generation. 3rd International Workshop on Protocol Test Systems, Washington, 1990.

[Sar 85] Sarikaya B. Bochmann G. Obtaining Normal Form Specification for Protocol, procedings COMNET'85. pag 601-612, Budapest, 1985.

[S&D 85] K.Sabnami. A.Dahbura. A New Technique for Generating Protocol Tests. AT&T Bell Laboratories, Murray Hill, N.J., USA. ACM 0-89791-164-4, 1985.

[S&L 89] D.P.SIDHU. T.K.Lung. Formal Methods for Protocol Testing: A Detailed Study. IEEE Transactions on Software Engineering, Vol 15, No 14, 1989.

[Ura 87] H.Ural. A Test Derivation Method For Protocol Conformance Testing. Proc Protocol Specification Testing and Verification VII (North Holland). pag 347-359 Elsevier Science Publishers B. V, 1987.

A DESCRIÇÃO DE UMA INTERFACE GRÁFICA E INTERATIVA PARA MATEMÁTICA COMPUTACIONAL

Álfio Ricardo Martini
Tiaraju Asmuz Diverio, Msc

Instituto de Informática e PGCC da UFRGS
Caixa Postal 1501 - 90 010 -Porto Alegre - RS
E-Mail : DIVERIO@SBU.UFRGS.ANRS.BR

RESUMO

Neste artigo é descrita a interface gráfica e interativa para matemática computacional (IMAC). Após anos de utilização de ambientes dirigidos por menus, apresentamos uma interface que utiliza uma linguagem de comando como forma de diálogo com o usuário. Primeiramente é descrito o modelo conceitual do usuário, ou seja, a compreensão conceitual que o usuário tem do programa. Então é introduzido o conceito de linguagens de comando, suas características e atributos desejáveis, a implicação direta destes conceitos no projeto da linguagem de comando de IMAC. Por fim, é apresentado o layout da tela da interface. No apêndice são apresentadas as principais operações (comandos) suportadas pela interface.

Palavras chaves : Linguagens de comando, interface usuário-computador, matemática computacional.

ABSTRACT

In this article an interactive graphics interface for researching in computational mathematics is described. After many years of using a menu-driven enviromnent, we present an interface that uses a command language as a model of user-computer dialog. Firstly, the conceptual user's model is described, i.e., the conceptual understanding that the user has about the program. Then the concept of command languages is introduced, its characteristics an desirable attributes, the direct implication of this concepts in the design of IMAC's command language. Finally, the interface screen layout is presented. In the appendix, the main operations (commands) supported by the interface are presented.

Key words : Command languages, user-computer interface, computational mathematics.

1. INTRODUÇÃO

Há alguns anos, no Instituto de Informática e Pós Graduação em Ciência da Computação da Universidade Federal do Rio Grande do Sul, tem-se estudado o problema do desenvolvimento de software de alta qualidade [1]. Como resultado deste estudo, diversas ferramentas foram implementadas, destacando-se **SINAI-16** [2] e **LEPMAC** [3].

Estes sistemas caracterizavam-se por um ambiente dirigido por menus e teclas aceleradoras. Como uma alternativa a este tipo de ambiente, surgiu a idéia do desenvolvimento de uma interface na qual o diálogo usuário-computador pudesse ser efetuado através de uma linguagem de comando. Desta forma, o usuário poderia ter acesso a diversos métodos do cálculo numérico computacional através da simples edição de comandos, os quais seriam, basicamente, os métodos disponíveis e seus respectivos argumentos (parâmetros). Além disso, este tipo de abordagem é mais interessante para usuários mais experientes como professores e pesquisadores da matemática computacional e interessados na resolução de problemas referentes a este assunto, tanto no que diz respeito a utilização do computador quanto ao tipo de tarefa a ser executada.

Como resposta a esta necessidade, foi projetada uma interface gráfica para matemática computacional (IMAC).

IMAC foi projetada para ser um ambiente integrado para trabalhar com uma enorme variedade de metodos computacionais como por exemplo, diferenciação e integração numérica, cálculo de zeros, resolução de sistemas de equações diferenciais , ajuste de curvas pelo critério dos mínimos quadrados, interpolação polinomial e resolução de sistemas de equações lineares.

Este artigo tem como objetivo principal descrever a estrutura de diálogo na interface de **IMAC**. Primeiramente é descrito o modelo conceitual do usuário, ou seja, a compreensão conceitual que o usuário tem do programa. Então é introduzido o conceito de linguagens de comando, suas características e atributos desejáveis e, a implicação direta destes conceitos no projeto da linguagem de comando de IMAC. Após, é apresentado o layout da tela de interface e, no apêndice são apresentadas as principais operações(comandos) suportadas pela interface.

2. O MODELO CONCEITUAL DO USUÁRIO EM IMAC

O modelo do usuário é o modelo conceitual formado pelo usuário da informação manipulada e dos processos aplicados sobre esta informação[4]. O modelo permite que o usuário desenvolva, com ou sem conhecimento da tecnologia de computadores, uma ampla compreensão sobre o que o programa está fazendo. Com a ajuda do modelo ele pode antecipar os efeitos de suas ações e pode desenvolver estratégias para operar o programa.

Uma maneira útil de se representar o modelo do usuário, para propósitos de discussão e documentação, é como um conjunto de **objetos** juntamente com um conjunto de **ações** que o usuário pode aplicar a estes objetos. Cada objeto é um item de informação sobre o qual o usuário tem algum controle, ele pode mostrá-lo, pedir informações sobre ele, destruí-lo e criar outros objetos em seu lugar.

IMAC tem quatro tipos fundamentais de objetos :

a) Expressoes Matemáticas (funções algébricas, transcendentes e polinomiais);

b) Tabelas de Pontos;

c) Sistemas de Equações Diferenciais Ordinárias;

d) Sistemas Matriciais;

2.1 Objetos e Ações

Expressões Matemáticas são criadas na interface de **IMAC** através da invocação de um comando específico (ver apêndice) seguido da expressão matemática desejada. É permitido trabalhar com até dez funções simultaneamente, ou seja, são sempre mantidas na memória as dez últimas funções criadas pelo usuário. Sempre que o usuário cria uma nova função ela se torna o que em **IMAC** é entendido como **função de trabalho.** É sobre esta função que o usuário pode realizar as seguintes ações:

- Integração numérica;
- Diferenciação numérica;
- Plotagem da função e suas derivadas (primeira e segunda);
- Cálculo de zeros;
- Ampliações;

A **integração numérica** de funções pode ser feita através dos métodos de Simpson, Trapézio, Romberg e Quadratura Gaussiana.

A **diferenciação numérica** pode ser feita de dois modos, **gráfico ou numérico.** No modo gráfico, o usuário pode "caminhar" com um cursor através do gráfico da função enquanto o valor da derivada (primeira e segunda) é automaticamente mostrado na janela de visualização da função. No modo numérico, o usuário edita o comando seguido do valor (parâmetro) numérico sobre o qual ele quer determinar o valor da derivada.

A plotagem das derivadas sempre se refere á **função de trabalho.** O **cálculo** de zeros de funções pode ser feito pelos métodos de **Newton, Jarrat, Pégasus e Miller.**

Ampliação de uma função significa a determinação de uma subjanela de visualização para a função de trabalho. A seleção dos valores para a determinação desta subjanela pode ser feita no modo **gráfico ou numérico.** No modo numérico, o usuário edita o

respectivo comando, seguido dos valores que determinam a subjanela, ou seja, **X1 Y1 X2 Y2.** No modo gráfico, a escolha destes valores pode ser feita pela construção da janela diretamente sobre a área a ser ampliada . Todos os valores iniciais utilizados nos métodos de integração (limites de integração) e cálculo de zeros (aproximaçào inicial) também podem ser determinados numericamente ou graficamente.

Além disso, a possibilidade de trabalhar com até dez funções ao mesmo tempo permite que se possa plotá-las todas na mesma janela de visualização (ver layout da interface).

Tabela de Pontos é o segundo tipo de objeto fundamental dentro da interface de **IMAC.** Tabelas de Pontos são formadas por uma série de dados (Xi,Yi) i = 0(1)n correspondentes aos valores de argumentos e valores de uma função f, tal que y = f(x), que são obtidos normalmente através de dados experimentais. Tabelas de Pontos podem ser criadas, editadas (deleção, alteração e inserção de pontos) gravadas e carregadas para a interface. Só pode existir uma tabela na interface de cada vez. Os valores são armazenados em memória, podem ser utilizados para ajuste de curvas pelo critério dos mínimos quadrados e interpolação polinomial. É permitido ajuste a curvas do tipo **linear, logarítmica, exponencial, potencial e polinomial** até sexto grau. A função ajustada é automaticamente considerada como **função de trabalho** e todas as operações descritas para as expressões matemáticas também podem ser efetuadas.

A interpolação de valores (x e y) pode ser feita através do método de **Newton por diferenças divididas.**

O terceiro tipo de objeto que pode ser trabalhado em **IMAC** são os **Sistemas de equações diferenciais ordinárias de primeira ordem.** Para a resolução de equações diferencias ordinárias de ordem > 1 (a ordem de uma equação diferencial ordinária é dado pela derivada de maior ordem presente na equação) é necessário transformá-la para um sistema equivalente de equações diferencias

de primeiro grau. O método de redução de uma ou várias equações diferenciais para um sistema de equações de ordem 1 é como segue, e está contido em [6].

a) Introduz-se novas variáveis para cada derivada até (m-1)-ésima ordem;

b) Substitui-se na equação diferencial original todo $y^{(m)}$ por cada uma das (m-1)-ésimas variáveis correspondentes e esta passa a ser a primeira equação do sistema.

c) Acrescenta-se a equação diferencial para a k-ésima variável, sendo esta a derivada primeira da (k-1)-ésima variável.

Por exemplo, a equação:

$$y''' + xy'' + 3y' - y = x^2 + y^2\sin(x)$$

após aplicarmos o método, gera o seguinte sistema :

$w' = x^2 + y^2\sin(x) - xw - 3z + y$

$y' = z$

$z' = w$

A solução de equações diferenciais em **IMAC** está limitada àquelas que podem ser modeladas pelo problema do **valor inicial.** Para cada equação diferencial do sistema, é necessário que haja uma condição inicial. Os sistemas de equações diferenciais possuem o mesmo comportamento semântico das tabelas de pontos, ou seja, possuem o mesmo tipo de operações básicas (criação, edição, gravação, etc...) e devem ser primeiramente criados para que depois sejam resolvidos. Novamente, só deve haver um sistema de equações diferenciais na interface de cada vez. Os métodos disponíveis são os de **Runge-Kutta de quarta ordem, Adams-Bashforth e Preditor-Corretor.**

O último tipo de objeto na interface de **IMAC** são os **sistemas matriciais.**Os sistemas matriciais envolvem as matrizes e os sistemas de equações lineares. O comportamento semântico dos

sistemas matriciais é idêntico ao da tabela de pontos e sistemas de equações diferenciais. As operações que envolvem a manipulação de matrizes são as de cálculo da matriz inversas, cálculo de normas e determinantes. Para a resolução de **sistemas de equações lineares**, os métodos disponíveis são os Métodos de Eliminação de Gauss com pivotamento simples, o método iterativo de Gauss-Seidel e o método compacto de Barnachevski.

Detalhes e complementação teórica a respeito dos métodos computacionais discutidos neste item podem ser encontradas em [5, 6, 7].

3 MODELO DE DIÁLOGO COM O USUÁRIO

3.1 Linguagens de comando

Linguagens de comando, as quais se originaram com os sistemas operacionais, são distinguíveis por seu imediatismo e por seu impacto em dispositivos de informação. O usuário edita um comando e observa o que acontece.

Linguagens de comando são distinguíveis de sistemas de seleção por MENUS pelo fato de que os usuários da primeira devem relembrar a notação e iniciar a ação. Usuários de sistemas orientados por MENUS recebem instruções e devem somente reconhecer e escolher entre um número limitado de alternativas visíveis. Eles respondem mais do que tomam iniciativa. Usuários de linguagens de comando são frequentemente exigidos a atingir feitos notáveis de memorização e digitação.

Ao projetarmos uma linguagem de comando, devemos considerar problemas referentes a sua sintaxe e a sua semântica. A sintaxe deve ser consistente e permitir fácil recuperação de erros sintáticos, ao mesmo tempo em que deve ser suficientemente compreensiva e poderosa para satisfazer as necessidades do usuário.

A parte semântica envolve o significado atribuído a cada construção sintática. A semântica de uma linguagem de comando relaciona-se, intimamente, com o **modelo do usuário.** Cada comando corresponde a uma ação, e os operandos (parâmetros) de cada comando são os objetos que a ação afeta.

Os objetos e ações do modelo fornecem um modelo inicial conveniente para o projeto da linguagem de comando. Eles definem, de uma forma abstrata, as operações que o usuário pode aplicar e os operandos nos quais elas podem ser aplicadas. A linguagem de comando de **IMAC** é uma representação concreta do modelo do usuário discutido anteriormente. Existem comandos para cada ação descrita, assim como muitos outros que aumentam o poder e a funcionalidade da interface.

Para a definição da sintaxe da linguagem, diversas qualidades desejáveis a uma boa linguagem de comando [8] foram consideradas, como por exemplo, facilidade de aprendizado, facilidade de uso, resistência a erros semânticos e consistência.

Facilidade de aprendizado pode ser entendida como facilidade de memorização. Facilidade de utilização normalmente significa facilidade de digitação. Resistência a erros semânticos refere-se a probabilidade de que um usuário possa digitar algo que ele não queira, e que assim mesmo é uma construção sintática válida dentro da linguagem. Quanto maior a probabilidade, maior a resistência. Consistência tem a ver com a presença de um estilo único na linguagem.

É sabido, através da experiência no projeto de linguagens de comando, que a presença de todas estas qualidades em uma mesma linguagem é praticamente impossível. Normalmente, uma grande dose de bom senso, experiência e esforço intelectual envolvem a escolha de uma em detrimento de outra característica.

No próximo item, serão discutidas as considerações principais envolvidas no projeto da linguagem de comando de **IMAC.**

3.2 Atributos de uma linguagem de comando e sua relação com a linguagem de comando de IMAC

A análise de uma linguagem de comando pode ser baseada em qualquer de seus atributos, mas apenas dois serão discutidos aqui; **estilo e estrutura**, pois são os que mais interessam dentro do contexto (modelo) para o qual a linguagem foi projetada.

Estilo refere-se ao fato de os comandos poderem ser extensos ou abreviados. Comandos extensos são menos eficientes quanto a digitação (utilização mais difícil), entretanto sua redundância faz com que a mesma seja menos sujeita a erros semânticos.

Estrutura se refere ao fato da relação entre os comandos e suas opções (argumentos) ser **linear** ou **hierárquica.** Uma estrutura linear fornece todos as funções da linguagem como ações distintas, ou seja, cada ação é representada por um comando. Uma estrutura hierárquica mostra um mesmo comando com várias opções. Uma estrutura linear normalmente é de utilização e memorização mais fácil do que uma estrutura hierárquica, ao passo que a última possui uma maior resistência a erros semânticos.

A linguagem de comando **IMAC** possui simultaneamente uma estrutura linear e hierárquica, sendo que, aproximadamente, metade dos comandos possui uma estrutura e o restante a outra (ver apêndice), além de permitir a edição de comandos extensos e abreviados.

Existem abreviações para praticamente todos os comandos da linguagem (ao menos para todos cujo comando extenso possui no mínimo quatro caracteres). A estratégia de abreviação adotada foi a de eliminação de vogais (a exceção do primeiro caracter) com

truncagem simples dos dois primeiros. Em alguns casos, em que a abreviação resultante foi considerada semanticamente insatisfatória ou então conflitante com alguma outra abreviação, foram utilizados o primeiro e o último caracteres.

Comando	Abrev.	Opção	Parâmetros
Cria	**Cr**	**Fun**	X^2 - COS(X*SIN(X^2))
SelFun	**Sl**	-	-
Plot	**Pl**	**Di**	-
Jarrat	**Jr**	-	0.4857 8
Diret	**Dr**	-	A:*.Exe

Figura 1. Exemplos da linguagem de comando

Na Figura 1 vemos um subconjunto de comandos sintaticamente válidos na linguagem de **IMAC**, juntamente com suas abreviações correspondentes. O caractere "!" serve para indicar uma opção entre o comando extenso ou sua abreviação. O primeiro comando faz com que a expressão matemática dada por **X^2 - COS(X*SIN(X^2))** torne-se a **função de trabalho.** O segundo permite que o usuário escolha outra **função de trabalho** através da seleção de uma nova função entre todas as outras já criadas anteriormente na interface. O terceiro comando causa a plotagem da primeira derivada da **função de trabalho.** O quarto permite o cálculo do zero da **função de trabalho** utilizando **0.4857** como aproximação inicial e **8** dígitos significativos exatos como precisão do resultado final. O últ mo permite mostrar o diretório do drive **A** correspondente ao parâmetro ***.Exe.**

Com base nas considerações feitas anteriormente, pode-se concluir que a linguagem de comando de **IMAC** é de fácil utilização, uma vez que além de existirem abreviações para praticamente todos os comandos, as regras utilizadas na estratégia de abreviação podem ser facilmente memorizadas pelo

usuário (somente duas). O estilo adotado na linguagem possibilita uma maior resistência a erros semânticos (comandos extensos) ao mesmo tempo que permite a acomodação de usuários mais experimentados que desejem executar suas tarefas com maior rapidez (comandos abreviados). A estrutura linear e hierárquica da linguagem facilita o aprendizado (linear) ao mesmo tempo em que contribui para aumentar a resistência a erros semânticos.

Na linguagem de comando de **IMAC**, um string representando um comando sintaticamente válido digitado pelo usuário é representado por um nome de comando **Nome**, seguido por um ou mais espaços em branco, **eb**, seguido por zero ou mais opções, seguido por zero ou mais parâmetros. Portanto, a sintaxe da linguagem de comandos de **IMAC** pode ser representado pelo seguinte conjunto de expressões regulares:

Comando = (String_Comando)$^{+}$ Delimitador
String_Comando = Nome eb (Opção eb)* (Param)*
Opção = Opção(eb Opção)*
Param = Nome (eb Nome)*
Nome = Letra(Letra|Dígito)*
Opção = Letra(Letra|Dígito)*
Letra = "A"..."Z","a"..."z"
Dígito = "0"..."9"
Delimitador = EOL
eb = " "

4.0 LAYOUT DA INTERFACE DE IMAC

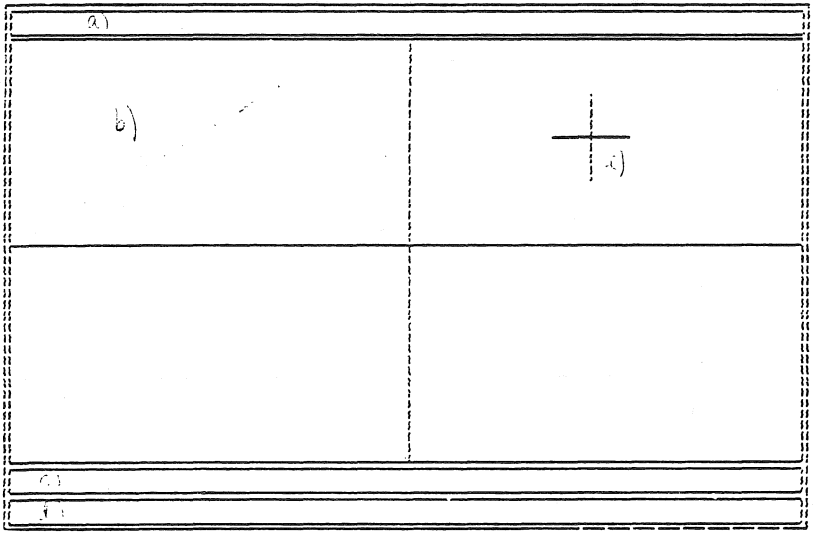


Figura 2. Layout da tela da interface

A Figura 2 apresenta o layout da tela da interface de **IMAC**. Nela vemos quatro áreas e o cursor. As áreas são: área da função atual, área de edição de comandos e área de teclas de funções especiais.

a) A área da função atual de trabalho serve para mostrar e expressão matemática da função que é correntemente a função de trabalho.

b) A área de visualização gráfica da função é utilizada para mostrar o gráfico da função atual de trabalho e de suas derivadas, assim também como a apresentação de janelas pop-up de resultados, diretórios , seleção de valores iniciais para métodos de integração e cálculo de zeros e outras atividades.

c) A área de edição de comandos é a área da interface em que o usuário interage com o computador através da edição de comandos.

d) A área de teclas de funções especiais serve para dois propósitos. Primeiramente como uma lembrança visual ao usuário de uma série de funções importantes atribuidas as teclas. Todas as funções são disponíveis também através da linguagem de comando, mas dada a sua importância e a frequência que as mesmas podem vir a serem utilizadas, a atribuição direta destas funções a algumas teclas facilitam e motivam a sua utilização, como por exemplo:

F1 = Help F2 = Imprimir F3 = Lista F4 = Ampliar

Em segundo lugar, a área de teclas de funções especiais serve para dar feedback ao usuário das atividades realizadas pelo programa, como por exemplo :

Gravando C:\IMAC\EXP001.EXP...
Integrando função pelo método de SIMPSON. Aguarde...

e) O cursor é utilizado para seleção de valores numéricos e determinação de subjanelas de visualização para plotagem de determinadas regiões do gráfico da **função de trabalho.**

5. CONCLUSÃO

Neste artigo, descrevemos uma interface gráfica e interativa para trabalho com métodos computacionais, cuja estrutura de diálogo usuário-computador é determinada através de uma linguagem de comando.

Além disso, mostramos que o projeto desta linguagem deve basear-se em dois aspectos fundamentais :

- Em um **modelo do usuário** bem definido, o que determina praticamente toda a definição da semântica (ações) desta linguagem.

- E que o projeto da sintaxe desta linguagem deve levar em conta fatores cognitivos (fatores humanos), ou seja, que ela tenha características tais que permitam um fácil aprendizado pelas pessoas que realmente vão utilizá-la e, como consequência, colabore para o sucessc do sistema.

6. REFERÊNCIAS

[1] DIVÉRIO, T. A. **Software Numérico Aplicativo e Instrucional.** Porto Alegre, PGCC da UFRGS, 1986 (dissertação de mestrado).

[2] DIVÉRIO, T.A. e Rombaldi, V. **SINAI-16. Um exemplo de software instrucional.** In: CNMAC, X, Gramado, set. 20-25, 1987.

[3] LEPMAC. Manual do usuário. Porto Alegre, PGCC da UFRGS, 1990.

[4] NEWMAN, M. W. and SPROULL F.R. **Principles of Interactive Computer Graphics.** McGraw-Hill, New York, 1979.

[5] RALSTON, A. **A First Course in Numerical Analisis.** McGraw-Hill, Tokyo, 1965.

[6] RICE, J.R. **Numerical Methods, Software and Analysis.** McGraw-Hill, New York, 1975.

[7] HORNBECK, R.W. **Numerical Methods.** Quantum Publishers, New York, 1975.

[8] HARDY, Jr. T. The Syntax of Interctive Command Languages: A framework for Design. **Software-Practice and Experience**, vol. 12, 67-75 (1982).

APÊNDICE

Os comandos suportados por IMAC

Este item apresenta a declaração de todos os comandos suportados na interface de **IMAC**. Os comandos e suas abreviações aparecem em negrito. Os parâmetros dos comandos devem ser sempre ser separados por no mínimo um espaco em branco. Para maior clareza e entendimento dos comandos, todos os parâmetros numéricos são acompanhados por seu tipo (inteiro ou real). O símbolo ":" é utilizado para separar um parâmetro numérico de seu respectivo tipo . A linguagem de comando de **IMAC** não é sensível ao caso, ou seja, pode-se misturar livremente caracteres maiúsculos e minúsculos. O símbolo reservado **Gr** da linguagem de comando indica que o valor numérico correspondente pode ser selecionado via cursor. Os comandos serão divididos em comandos computacionais e comandos de uso geral.

Comandos Computacionais

Comandos para integração numérica de funções

Rm | **Romberg** X1:Real|Gr X2:Real|Gr Digse:Integer
Sm | **Simpson** X1:Real|Gr X2:Real|Gr Digse:Integer
Tr | **Trapezio** X1:Real|Gr X2:Real|Gr Digse:Integer
Qd | **QuadGauss** X1:Real|Gr X2:Real|Gr Digse:Integer

Onde :

X1,X2 : Parâmetros que definem o limite de integração
Digse : Dígitos significativos exatos requeridos na operação

Comandos para o cálculo de zeros de funções

Nw | **Newton** X1:Real|Gr Digse:Integer
Pg | **Pegasus** X1:Real|Gr X2:Real|Gr Digse:Integer
Ml | **Miller** X1:Real|Gr X2:Real|Gr X3:Real|Gr Digse:Integer
Jr | **Jarrat** X1:Real|Gr Digse:Integer

Onde, X1,X2,X3 : Valores que fornecem a aproximação inicial para o método em questão.

Comando para interpolação de valores dada uma tabela de pontos

Nt | **NewtonInt** X|Y Valor|Gr

Onde, X,Y : Representa o parâmetro que se quer interpolar Valor : Representa o valor do parâmetro a ser interpolado

Comando para a resolução de equações diferenciais ordinárias

Edo X1:Real X2:Real h:Real positivo Método

Onde, X1,X2 : Intervalo sobre o qual se quer procurar a solução da equação diferencial

h : Passo

Método : Método a ser utilizado na resolução da equação diferencial. Os métodos possíveis são os de Runge-Kutta4, Adams-Bashforth e Preditor e Corretor.

Para invocar este comando é preciso, inicialmente, ter criado um sistema de equações diferenciais com o comando **CriaEdo** Grau.

Comandos para resolução de sistemas de equações lineares

Gs | **Gauss** -> Permite a resolução do sistema de equações lineares correntemente na memória através do método de eliminação de Gauss.

Gl | **Gauss-Seidel** -> Permite a resolução do sistema de equações lineares correntemente na memória através do método iterativo de Gauss-Seidel.

Br | **Barnac** -> Permite a resolução do sistema de equações lineares correntemente na memória através do método de Barnachevsky.

Sl ! **Salva** Opção Nome_Arquivo

Onde Opção -> **Edo** ! **Tab** ! **Mat** !**Sis**!**Fun**

Edo -> Permite salvar um sistema de equações diferenciais previamente criado com o comando **Cria Edo**. O sistema é gravado com o nome de Nome_Arquivo. A extensão **.EDO** é automaticamente verificada pelo programa.

Tab -> Permite salvar uma tabela de pontos previamente criada com o comando **Cria Tab**. A tabela é gravada com o nome de Nome_Arquivo. A extensão **.TAB** é automaticamente verificada pelo programa.

Mat -> Permite salvar uma matriz previamente criada com o comando **Cria Mat**. A matriz é gravada com o nome de Nome_Arquivo. A extensão **.MAT** é automaticamente verificada pelo programa.

Sis -> Permite salvar um sistema de equaçoes lineares previamente criado com o comando **Cria Sis**. O sistema é gravado com o nome de Nome_Arquivo. A extensão **.SIS** é automaticamente verificada pelo programa.

Fun -> Permite salvar uma função de trabalho previamente criada com o comando **Cria Fun**. A função é gravada com o nome de Nome_Arquivo. A extensão **.EXP** é automaticamente verificada pelo programa.

Co ! **Car Edo** Nome_Arquivo -> Permite carregar um sistema de equações diferenciais para a memória que tenha sido gravado com o comando **SalvaEdo**.

Cr ! **Carrega** Opção Nome_Arquivo -> Permite carregar a memória do programa um arquivo previamente gerado pelo sistema.

Onde Opção -> **Tab** ! **Edo** ! **Fun** ! **Sis** ! **Mat**

Ed ! **Edit** Opção -> Permite editar um **objeto** da interface de **IMAC** definido pelo parâmetro opção.

Comandos de Uso Geral

Lm | **Limpa** -> Este comando permite limpar a área de visualização gráfica da função.

Cr | **Cria** Opção

Onde Opção -> **Tab** | **Fun** | **Edo** | **Sis**|**Mat**

Tab N:Integer -> Esta opção permite criar uma tabela de pontos (pares ordenados) onde o usuário pode efetuar operações de ajuste e interpolação. O valor de N deve ser <= 50.

Fun função -> Esta opção possibilita a criação de uma função de trabalho. O parâmetro função significa uma expressão matemática. A invocação deste comando também causa a plotagem automática da função na área de visualização gráfica da função.

Edo N:Integer -> Permite a criação de um sistema de equações diferenciais para ser resolvido pelo comando **Edo**. O número de equações do sistema está limitado a um máximo de 10, ou seja, é possível resolver equações diferenciais ordinárias de até décima ordem. Deve-se entrar uma condição inicial para cada equação após a respectiva equação ter sido digitada. O símbolo ";" serve como delimitador da equação e de sua respectiva condição inicial.

Exemplo :

$$y' = 3x + 4 \; ; \; y(0) = 2$$

Mat Ordem -> Permite e entrada de uma matriz de ordem dada pelo parâmetro **Ordem**. A ordem máxima permitida é de 20.

Sis Ordem -> Permite a entrada de um sistema de equações lineares cuja ordem é dada pelo parâmetro **Ordem**. A ordem máxima permitida é de 20.

Sn | **Salva Fun** Nome_arquivo -> Este comando permite gravar a função atual de trabalho em disco. Nome_arquivo pode também indicar o drive e o path. A extensão **.FUN** é automaticamente verificada pelo programa.

Onde Opção -> **Tab** | **Edo** | **Fun** | **Sis** | **Mat**

Sn | **SelFun** -> Este comando permite ao usuário escolher uma nova função de trabalho a partir de uma janela que mostra as últimas (até um máximo de dez) funções criadas na interface.

Pl | **Plot** D1 | D2 | Fun -> Este comando permite ao usuário plotar a função atual de trabalho e/ou sua derivada primeira e segunda.

Cl | **Calc** D1 | D2 Gr | Num_Real -> Permite calcular o valor da derivada primeira ou segunda de uma **função de trabalho.** Este cálculo pode ser feito através da especificação de um número real ou então de maneira gráfica, onde a medida que o usuário "caminha" com o cursor sobre a função, o valor da derivada (primeira ou segunda) é automaticamente mostrado.

Im | **Imprime** Lan | Port -> Este comando permite ao usuário fazer uma cópia gráfica da tela no modo **Landscape** ou no modo **Portrait.** No modo Portrait o eixo C da tela é plotado ao longo da largura da folha do papel e o eixo X ao longo do comprimento da folha do papel, produzindo uma imagem que ocupa metade da folha. Por outro lado, no modo Landscape, o eixo X da tela é plotado de acordo com o comprimento da folha e o eixo Y de acordo com a largura, produzindo, desta forma, uma imagem do tamanho de uma folha.

Cn | **Const** -> Este comando permite ao usuário criar uma tabela de constantes, que podem ser utilizadas tanto nas funções quanto nos sistemas de equações diferenciais. Cada constante tem um nome (máximo de dez caracteres) e um valor associado. A invocação deste comando causa o aparecimento de uma janela pop-up que contém todas as constantes já definidas pelo usuário. É permitido a alteração, deleção e inserção de constantes. Entretanto, existem algumas constantes que são pré-definidas pela interface, e que não podem ser alteradas nem deletadas.

Ls ! **Lista** -> Causa o aparecimento de uma janela que contém os últimos comandos (até um máximo de dez) editados pelo usuário e que são sintaticamente válidos de acordo com a linguagem de comando. A seleção de algum comando retira a janela de apresentação dos comandos e deixa o comando selecionado na área de edição, pronto para modificações ou até invocação imediata.

Hp ! **Help** -> Causa o aparecimento de uma janela que permite (através de scrolling vertical) a visualização de todos os comandos da interface. A seleção de algum item causa o aparecimento de uma outra janela (em offset em relação a anterior) com uma breve descrição do comando e também a respeito de sua declaração.

Al ! **AltDir** S:String -> Permite ao usuário alterar o diretório corrente para um path especificado por S. Se S especificar um drive, então o drive corrente também é selecionado.

Dr ! **Diret** Path + Máscara -> Permite visualizar todos os arquivos que coincidam com o argumento Path + Máscara. É permitido a navegação vertical e horizontal entre os arquivos, de tal maneira que o usuário possa selecionar algum arquivo do sistema (funções, tabela de pontos, sistema de equações diferenciais, sistemas de equações lineares e matrizes) para a memória.

Ex ! **Eixo** X!Y On!Off -> Este comando permite que retire o eixo X e/ou o eixo Y da janela de visualização da função.

Es ! **Escala** X!Y V1:Real V2:Real -> Este comando permite que o usuário altere os valores da escala X e Y para os correspondentes valores numéricos de V1 e V2.

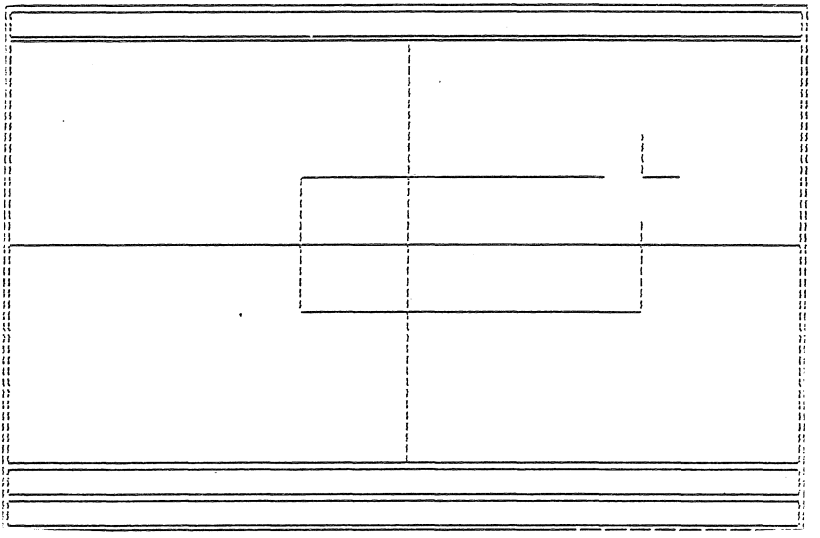


Figura 3. Criação de uma subjanela da função de trabalho

Am | **Amplia** Gr | X1:Real Y1:Real X2:Real Y2:Real -> Este comando permite ao usuário selecionar uma subjanela de visualização (Figura 3) para a função atual de trabalho. As coordenadas desta subjanela podem ser determinadas diretamente pelos parâmetros X1,Y1,X2 e Y2. As coordenadas também podem ser determinadas graficamente através de um cursor que "caminha" sobre a função. O símbolo **Gr** identifica esta opção.

Ms | **MostraEsc** -> Este comando causa o aparecimento de uma janela com os valores das coordenadas atuais da janela de visualização da função.

MODELAGEM DE SISTEMAS DE MANUFATURA : UMA ABORDAGEM ORIENTADA A OBJETOS

Luis Fernando Friedrich
Universidade Federal de Santa Catarina
Departamento de Computação

ABSTRACT

The paper shows an Object-Oriented Approach to Software Modelling of Manufacturing Systems. This approach enable the structure and behavior specication of manufacturing systems. The structure is represented by software objects and the relationship between these objects are defined by system activities. The behavior description is built up through activities that the manufacturing system realize.

RESUMO

O artigo apresenta uma Abordagem Orientada a Objetos para a Modelagem de Sistemas de Manufatura. A abordagem proposta permite a especificacão da estrutura assim como do comportamento de um sistema de manufatura. A estrutura (parte operativa) é representada pelos objetos físicos componentes do sistema. O comportamento é descrito pelas atividades realizadas e que causam o fluxo dos objetos através do sistema. Os objetos físicos são modelados por objetos de software e, o relacionamento entre estes objetos definem as atividades dos objetos de mais alto nível, caracterizando assim uma hierarquia de especificação dos objetos componentes de um sistema de manufatura.

INTRODUÇÃO

A crescente utilização de máquinas de comando numérico, controladores lógicos programáveis, robos industriais, computadores e processadores de uso geral e sistemas gráficos, tem trazido nos últimos anos grandes transformações à indústria.

A tendência das inovações tecnológicas não se encontra apenas na sofisticação e aprimoramento destes elementos, como também na união destes elementos em sistemas integrados e com alto grau de flexibilidade.

Estes fatores tem como resultado um aumento no grau de automatização , assim como na complexidade, tanto a nível de projeto como de operação. Isto faz com que seja necessária a existência de metodologias de desenvolvimento com o propósito de auxiliar na especificação e projeto de sistemas que integram elementos como os citados anteriormente.

A necessidade de metodologias e linguagens que se apliquem ao desenvolvimento de sistemas de manufatura, com atenção especial aos aspectos de especificação e prototipação, constitui-se em uma das motivações deste trabalho. Mas a idéia mais importante que deve ser mantida durante o desenvolvimento do trabalho é a busca de uma metodologia simples que, além de atender as necessidades existentes, possa ser totalmente compreendida e manipulada pelos técnicos de cada empresa, sejam eles especialistas em informática sem conhecimento de engenharia ou, especialistas em engenharia sem conhecimento de informática.

SISTEMAS DE MANUFATURA

De um modo geral o termo *manufatura* tem sido utilizado para definir o conjunto de todas as atividades envolvidas na fabricação de um produto, desde o recebimento da definição do mesmo até a sua realização final. Um Sistema de Manufatura (*SM*) é a união de todos os elementos utilizados nestas atividades dentro de uma empresa.

Um *SM* pode ser considerado como sendo composto por vários sub-sistemas como células de montagem, robôs, máquinas operatrizes, centros de trabalho, transportadores, etc., de forma que cada um destes sub-sistemas executem uma atividade produtiva, ou seja, uma transformação física (*usinagem*, *montagem*, *etc*), ou um deslocamento (*carregamento*,*descarregamento*, *etc*).

A caracterização de um processo de manufatura se dá através do fluxo de itens (*material*, *recurso*, *etc*) que passa sequencialmente pelos sub-sistemas, onde atividades produtivas são realizadas [*MIY89*]. Dois pontos são importantes para caracterizar um processo de manufatura : as *atividades* realizadas e as *condições* de realização das mesmas. O relacionamento entre as atividades e condições são de natureza discreta, e por isto estes sistemas são denominados como Sistemas Produtivos de Eventos Discretos (*SPEDs*).

Elementos dos Sistemas de Manufatura

No que diz respeito à representação de sistemas de manufatura [*WAL89*], alguns elementos recebem um destaque especial

devido a sua importância na caracterização deste tipo de sistemas. Estes elementos representam os componentes de um sistema de manufatura, são eles :

RECURSOS:

Os *recursos* representam os equipamentos físicos necessários a realização de um processo de manufatura . Estes recursos podem ser considerados em vários níveis de abstração. . Os recursos de um modo geral realizam atividades dentro do sistema de manufatura, e podem ser caracterizados por atributos que podem ser :

- atributos fixos definindo a natureza do recurso e suas características .
- atributos variáveis que podem evoluir no decorrer do tempo (estado de uma máquina).

PEÇAS:

Uma *peça* ou *material* representa qualquer item identificável dentro de um sistema de manufatura. Peça é todo material físico que participa do processo de produção como uma matéria prima, uma peça propriamente dita, sub-montagem ou produto final. Da mesma forma que um recurso, uma peça é caracterizada pelos seus atributos e também pode ser vista em níveis de abstração diferentes.

ATIVIDADES:

Uma *atividade* representa a modificação do valor de um atributo (característica) de uma peça, como por exemplo a sua forma, localização, temperatura, etc. As atividades requerem, além dos seus insumos, diversos recursos, tais como um centro de trabalho, ferramentas, mão de obra, para serem realizadas. Uma atividade

pode ser definida em vários níveis de abstração, dependendo do nível de controle/planejamento que ela está sendo especificada. Por exemplo, uma atividade **carregar peça na máquina** em um centro de usinagem pode ser decomposta em várias outras atividades, envolvendo outros recursos como por exemplo uma esteira, um robô e uma máquina-ferramenta :

esteira <= **parar**

robô <= **carregar peça** da esteira

máquina-ferramenta <= **receber peça** do robô.

Características de Modelagem de Sistemas de Manufatura

Aqui serão apresentadas algumas características importantes para a modelagem de sistemas de manufatura, a serem aplicadas aos níveis de supervisão global, supervisão de centros de trabalho e controle local. Atualmente algumas metodologias tem sido usadas, fluxogramas, diagramas de estado e diagramas de tempo, para especificação e projeto de sistemas de manufatura. Um ponto importante para uma metodologia de desenvolvimento é a descrição do comportamento dos sistemas de maneira completa, consistente, concisa, sem ambigüidades e precisa.

De acordo com os níveis de controle, um sistema de manufatura pode ser considerado ou não como um sistema de eventos discretos, isto é, pode ser modelado por *eventos* e *atividades* (operações). Para os três níveis mais baixos da hierarquia a modelagem de sistemas de manufatura pode ser feita usando a noção de sistemas de eventos discretos.

Os sistemas de manufatura são complexos e tendem a aumentar o seu nível de complexidade assim que novas funções são a eles incorporadas. Desta forma a modelagem deste tipo de sistemas deve permitir representar o sistema em diferentes níveis de abstração e facilitar a sua decomposição em subsistemas funcionais, com uma definição das relações entre estes subsistemas.

A complexidade inerente aos sistemas de manufatura faz com que este tipo de sistema não possa ser modelado a partir de um modelo que utilize o conceito de estado global, porque a partir de um estado, são muitos os eventos que podem provocar alterações no estado do sistema. È importante a representação dos diversos aspectos estáticos e dinâmicos do sistema, através de uma forma precisa e uniforme.

Uma característica importante nos sistemas de manufatura é que as atividades realizadas podem acontecer (ser executadas) em paralelo, total ou parcialmente independentes. Isto faz com que problemas de sincronização entre atividades e a competição pelos recursos que podem ser acessados por mais de uma atividade, devam ser modelados. Desta forma a metodologia de modelagem a ser usada deve facilitar a decomposição das atividades em outras atividades total ou parcialmente independentes, permitir a expresssão clara da sincronização entre as atividades e da competição ao recursos comuns.

Uma outra consideração importante é que este tipo de sistema envolve geralmente especialistas de diferentes áreas, e para que todos contribuam no desenvolvimento do sistema é

importante que a metodologia de modelagem seja de fácil aprendizado e compreensão, além de possuir boa legibilidade, preferencialmente com uma representação gráfica simples.

Resumindo o que foi discutido acima, alguns problemas são relevantes e devem ser considerados para a representação de sistemas de manufatura :

a) a representação do sistema em diferentes níveis de abstração;

b) a decomposição do sistema em subsistemas funcionais, com uma definição clara das relações entre eles;

c) a expressão da sincronização entre atividades;

d) a expressão da concorrência entre atividades;

e) a possibilidade de execução do modelo.

ABORDAGEM PROPOSTA

Similarmente aos Sistemas de Processamento de Dados, os Sistemas de Manufatura passam por duas fases principais durante o seu *ciclo de vida*, a fase de desenvolvimento (projeto) e a fase de operação e manutenção. A grande complexidade e o alto custo de implantação que envolve os Sistemas de Manufatura implicam na necessidade de modelos de representação que, através do uso de formalismos de representação convenientes (próprios), que suportem os objetivos da modelagem, permitam a validação das soluções propostas através de análise ou de simulação.

A proposta apresenta uma abordagem onde o processo de

manufatura é visto como sendo composto por duas partes principais, a parte operativa e a parte de controle, a figura 1 ilustra esta visão.

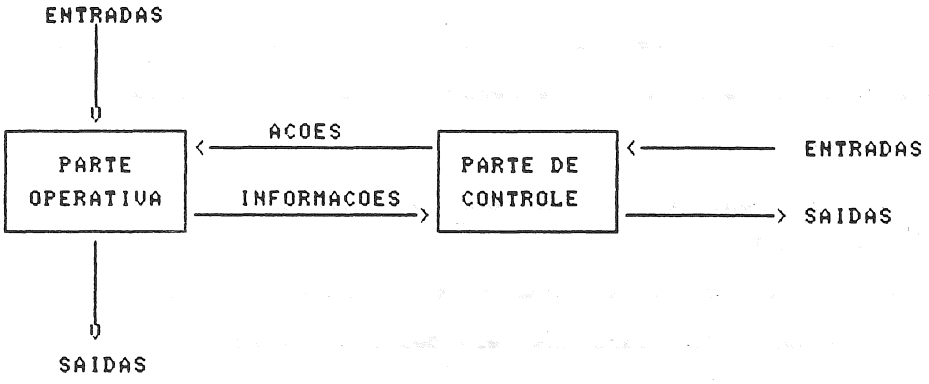


Fig. 3.1 Decomposicao do processo de manufatura

- a *parte operativa* engloba o conjunto de recursos físicos (máquinas, equipamentos, mão de obra) que são utilizados para armazenar ou transformar as matérias primas (pecas).

- a *parte de controle* assegura o sequenciamento e condiciona a execução de operações na parte operativa. A cada estado da parte de controle é associado um conjunto de ações executadas na parte operativa [*TEL84*].

A abordagem proposta pretende descrever a execução das atividades de manufatura em termos de *entidades de manufatura* e *atividades de manufatura*. As entidades de manufatura representam a parte operativa (estrutura), enquanto que as atividades de manufatura representam a parte de controle (comportamento) dos SM. A abordagem propõe um modelo de software para representação dos sistemas de manufatura, onde o sistema é composto de objetos

(peças, equipamentos) e relações entre os objetos (atividades), que são definidas (sequenciadas) pela parte de controle.

A proposta introduz uma metodologia de representação onde os sistemas de manufatura são descritos levando-se em consideração a sua estrutura e o seu comportamento dinâmico. A estrutura é descrita a partir de seus componentes e suas características que definem atributos e operações individualmente. O comportamento dinâmico é caracterizado por atividades e fluxo de "coisas" (materiais, informações, recursos). As atividades nos sistemas de manufatura podem representar diferentes níveis conceituais, sugerindo assim uma estrutura hierárquica na descrição das atividades.

A seguir é mostrado como a parte operativa e a parte de controle são representadas e de que forma o modelo gerado a partir da especificação do usuário pode ser analisado. A proposta apresentada considera a criação de um ambiente de desenvolvimento, a nível de protótipo, de sistemas de manufatura com base na metodologia de representação introduzida.

Representação da Parte Operativa

Com o objetivo de representar os componentes de sistemas de manufatura é proposta a utilização das técnicas de orientação a objetos [*BOOS86*] [*COX86*]. Objetos são usados para modelar entidades de manufatura como: recursos (equipamentos, mão de obra), materiais (peças, matéria prima), etc., que caracterizam entidades físicas de SM.

A modelagem de software é um processo de criação de uma representação ou modelo do sistema considerado, em um sistema de computação. Este modelo é geralmente criado a partir da descrição dos objetos no sistema e das interações dos objetos. Um dos problemas clássicos para uma abordagem genérica para a modelagem de sistemas de manufatura é a necessidade de incorporar abstração em múltiplos níveis. Esta necessidade existe pelo fato de que pessoas em diferentes níveis na hierarquia da organização tendem a ter uma visão dos componentes do sistema em diferentes níveis de abstração. Os sistemas orientados a objetos permitem que estes componentes possam ser tratados em diferentes níveis de abstração, possibilitando assim múltiplas visões da situação de uma maneira completa e consistente.

Estudos de problemas de implementação de sistemas de manufatura [*BES85*], tem recomendado uma abordagem incremental para a implementação. No paradigma de orientação a objetos duas características principais, *modularidade dos objetos* e *conceito de herança*, fazem com que esta abordagem seja possível.

Alguns autores [*ENG85*] [*TAM84*] tem identificado vários requisitos para modelagem de software em sistemas de manufatura. Estes requisitos em sua maioria incluem interfaces de usuário e interfaces de desenvolvimento de software. Os ambientes de desenvolvimento orientados a objetos na sua maioria incorporam grande parte dos requisitos identificados.

As entidades de manufatura, sejam elas físicas ou conceituais, são modeladas como objetos de software que apresentam um conjunto de características (atributos) e um conjunto de

operações (métodos) e são usadas para descrever a execução de atividades de um sistema de manufatura. Por exemplo, são entidades de manufatura : *informação*, *materiais*, *recursos*. *Informação* pode ser pedidos, informações técnicas, etc. *Materiais* incluem todas as 'coisas' físicas que entram no produto durante o processo de manufatura (matéria prima, peças, etc). *Recursos* representam todos os objetos físicos que realizam o processo de manufatura (equipamentos, ferramentas, pessoal, etc).

A identificação dos objetos é feita, pelos usuários, levando em consideração as entidades de manufatura que representam os tipos de objetos. Os objetos são definidos a partir da especificação de suas características como estrutura, operações e restrições. Objetos que apresentam características iguais são reunidos em classes. A estrutura dos objetos de uma classe e um conjunto de atributos que possuem um nome e um tipo, o tipo dos atributos pode ser uma constante ou uma referência a outra classe de objetos. No caso de um atributo do tipo referência o valor do mesmo é o nome de um objeto da classe referida e assim, os operadores que se aplicam neste caso permitem a chamada de uma operação do objeto. O valor de um objeto é o conjunto dos valores de seus atributos.

Um objeto apresenta operações e atributos públicos, que podem ser utilizados por outros objetos, e privados , que são ignorados pelos outros objetos e utilizados exclusivamente pelas operações do próprio objeto. Os serviços que um objeto coloca a disposição de seus clientes são, desta forma, a consulta de atributos publicos e a chamada de operações públicas.

O paradigma de orientação a objetos é considerado aqui como modelo para a representação da estrutura dos sistemas de manufatura, permitindo a especificação das entidades componentes e participantes de um SM. A modelagem orientada a objetos torna possível uma correspondência natural e direta entre o mundo da manufatura e seu modelo.

Representação da Parte de Controle

A parte de controle é responsável pela coordenação e especifica como os elementos componentes de um sistema de manufatura participam nas atividades realizadas. Na representação da parte de controle são consideradas as atividades que definem o processo de manufatura (sistema).

As atividades são consideradas, no modelo apresentado, como sendo transições simples (representando eventos), ou através da utilização de uma representação agregada que associa a uma transição a sequência *início de atividade / atividade / fim de atividade*. A representação agregada é mais apropriada no sentido de que ela detalha os eventos importantes de uma atividade (início e fim), e permite que se possa associar à atividade o tempo de realização da mesma (tempo padrão).

As atividades envolvem um conjunto de entidades que são necessárias para a sua realização, ou seja, são as condições que permitem que uma determinada atividade seja realizada. O modelo aqui considerado representa a sequência de realização das atividades de um SM, e as condições (recursos,materiais,etc.)

necessárias para a realização das mesmas. Cada atividade determina também um conjunto de ações que são executadas em função da realização da atividade. Estas ações são executadas pelos objetos componentes do SM, sendo que as ações correspondem a operações (atividades) definidas para estes objetos.

Os lugares na rede representam os objetos que participam das atividades, onde as marcas são individualizadas, ou seja, caracterizam as instâncias de um objeto. Desta forma uma atividade pode ser representada por um conjunto de lugares que definem os objetos que participam da sua realização, uma transição (evento) de início da atividade, um lugar que modela a atividade e uma transição de fim da atividade. Os resultados da execução de uma atividade de manufatura são colocados nos lugares de saída da atividade, seja através da criação de novos objetos ou da mudança de estado dos objetos.

A modelagem com as redes de Petri permite descrever aspectos importantes nos sistemas de manufatura como o paralelismo (duas atividades independentes), a competição por um recurso, a escolha entre alternativas, e a sincronização entre atividades. As redes permitem ainda uma boa visualização do fluxo de recursos e materiais, assim como informações, existente entre as atividades de manufatura.

A partir da especificação da rede que representa a parte de controle do sistema de manufatura, a idéia é criar um modelo de simulação onde a interpretação da rede define as atividades a ser realizadas a cada ciclo e as ações que os objetos participantes devem realizar. Neste caso cada atividade tem associada

pré-condições e pós-condições que determinam as restrições ações.

Os objetos considerados nos sistemas de manufatura podem ser complexos (agregados), ou primitivos. Nos objetos primitivos a rede que descreve a parte de controle representa o comportamento do objeto a nível de atividades que ele realiza.

O modelo proposto para a representação da parte de controle dos sistemas de manufatura permite a descrição das atividades e dos recursos utilizados, sejam eles equipamentos ou materiais. As atividades seguem uma hierarquia onde o seu conteúdo pode ser detalhado, ou seja, uma atividade pode ser representada por várias sub-atividades inter-relacionadas num nível inferior.

Exemplo

O exemplo a seguir descrito envolve um sistema composto por tres máquinas que realizam atividades e um conjunto de depósitos intermediários que armazenam peças, sobre as quais são realizadas operações. O processo é descrito da seguinte forma : quando os depósitos 1 e 2 contém pelo menos uma peça cada um a atividade 1 pode ser iniciada, sendo realizada pela máquina 1; o término da atividade implica na colocação de um produto no depósito 3; quando existe um produto no depósito 3 pode-se iniciar a atividade 2 ou a atividade 3.

Os objetos considerados no exemplo representam as peças e as máquinas (parte operativa) componentes do sistema, enquanto a rede de Petri representa o sequenciamento e as condições de

realização (parte de controle) das atividades do sistema. A figura 2 mostra os objetos (classes) que compõe o processo de manufatura.

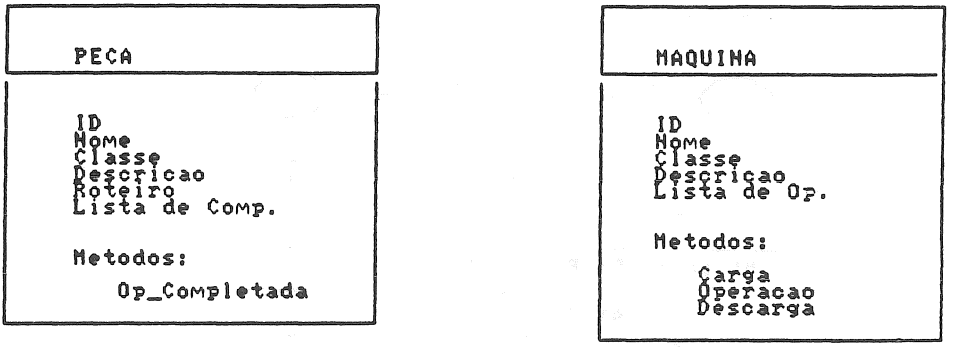


Fig. 2 Representacao dos objetos peca e maquina

A figura 3 mostra a representação do comportamento do processo (parte de controle). A representação abrange o fluxo de objetos (materiais,recursos) assim como de que forma o processo interage com os objetos componentes do sistema.

Neste caso os lugares representam as classes de objetos (L1,L2,L4 representam a classe peça), e a realização das atividades (AT_1,AT_2,AT_3 representam as atividades do processo). As transições representam os eventos de início e fim de uma atividade, e a elas são associadas pré-condições e ações. As pré-condições podem determinar o valor dos atributos dos objetos a serem selecionados. As ações mostram de que forma os objetos participam da realização das atividades deste nível. As ações são atividades realizadas pelos objetos componentes do nível de

controle que esta sendo descrito, por exemplo :

M1.carga (P1,P2).

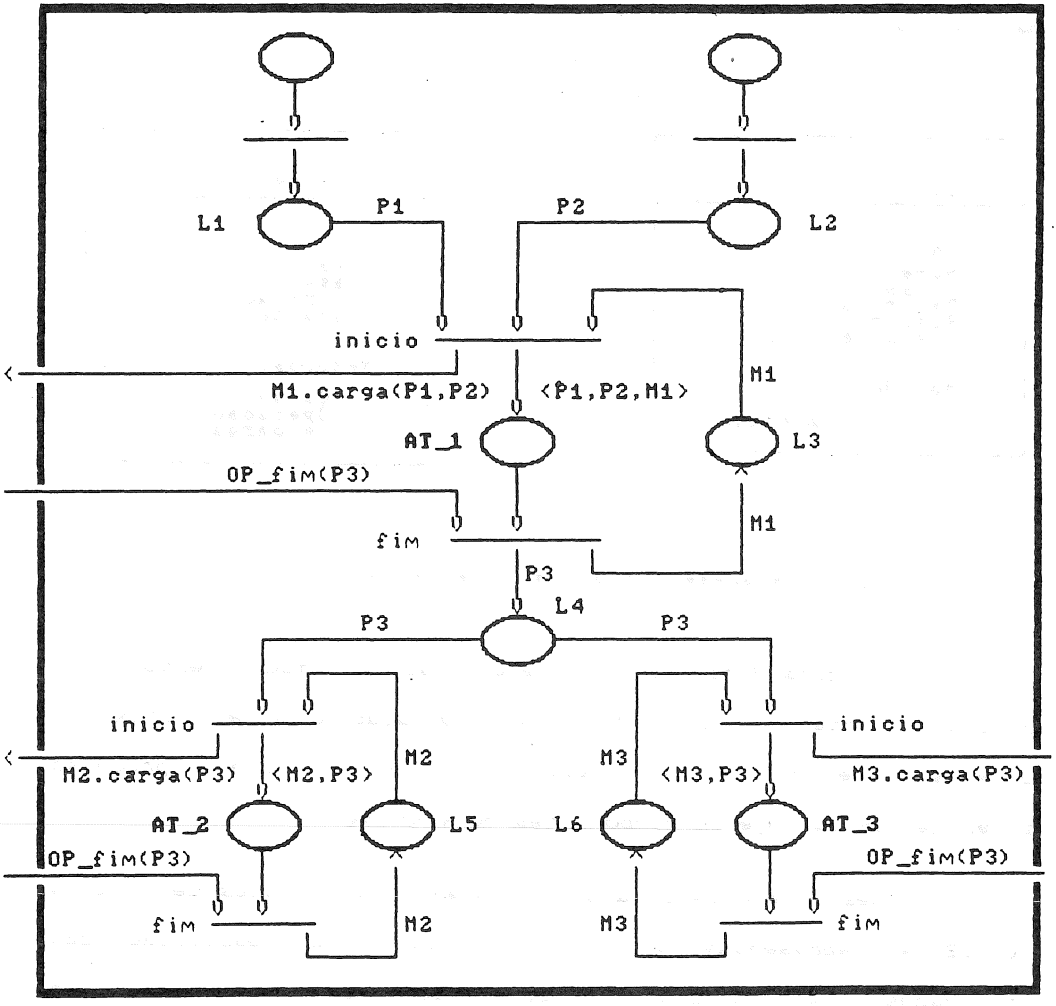


Fig. 3 Representacao da parte de controle

significa que o objeto *M1* participa da atividade AT_1, através da execução da sua atividade *carga*. São definidos ainda os atributos necessários para a realização da atividade, *P1 P2*, que por sua vez podem ser referências (classes de objetos) ou constantes.

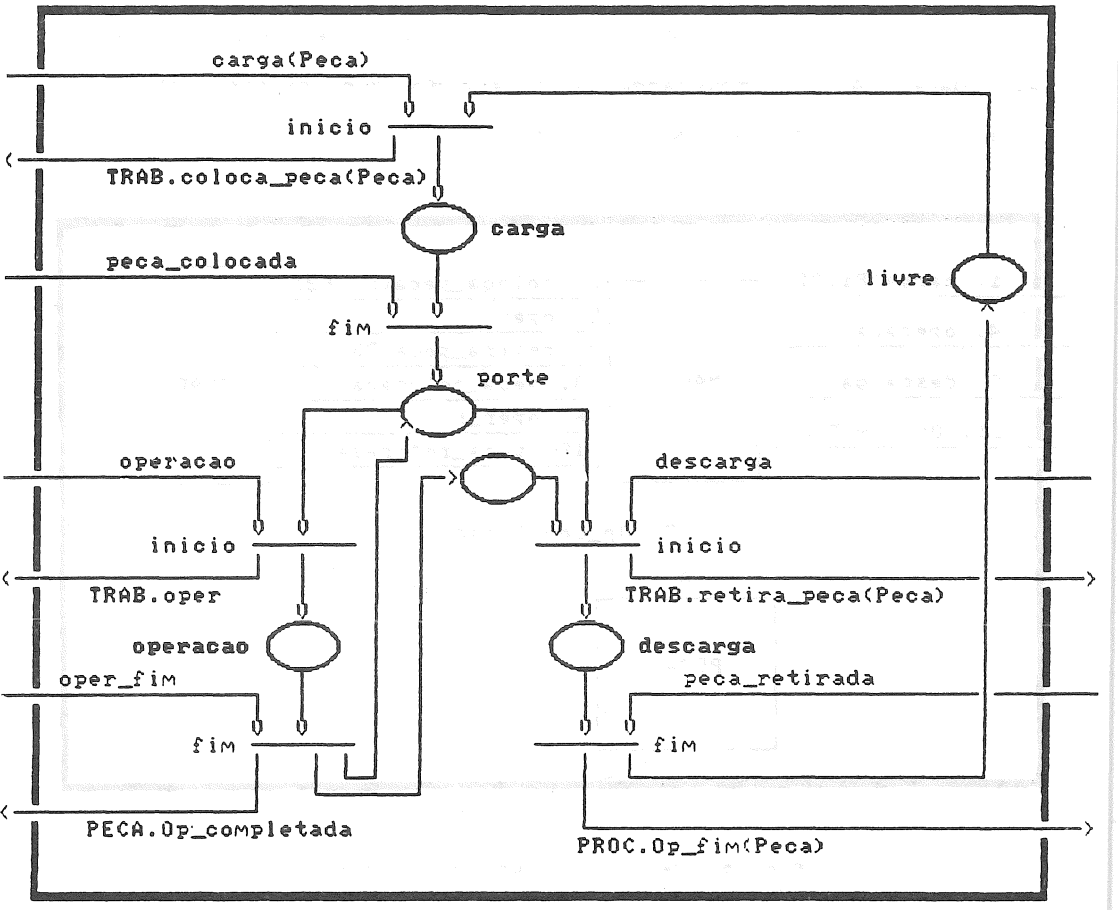


Fig. 4 Parte de Controle do objeto Maquina

A figura 4 mostra a representação da parte de controle (comportamento) da classe de objetos *Máquina*. Neste nível de descrição aparece um novo objeto TRAB que não é visto pelo processo descrito na figura 3.4, e que participa das atividades do objeto *Máquina*. A parte de controle do objeto máquina mostra como as atividades realizadas por este objeto se comportam. Os lugares *porte* e *livre* indicam estados internos do objeto, neste caso a ativação do objeto por uma mensagem depende do seu estado interno.

A figura 5 apresenta o cenário que descreve uma atividade (AT_1), mostrando a interação que existe entre os objetos participantes desta atividade.

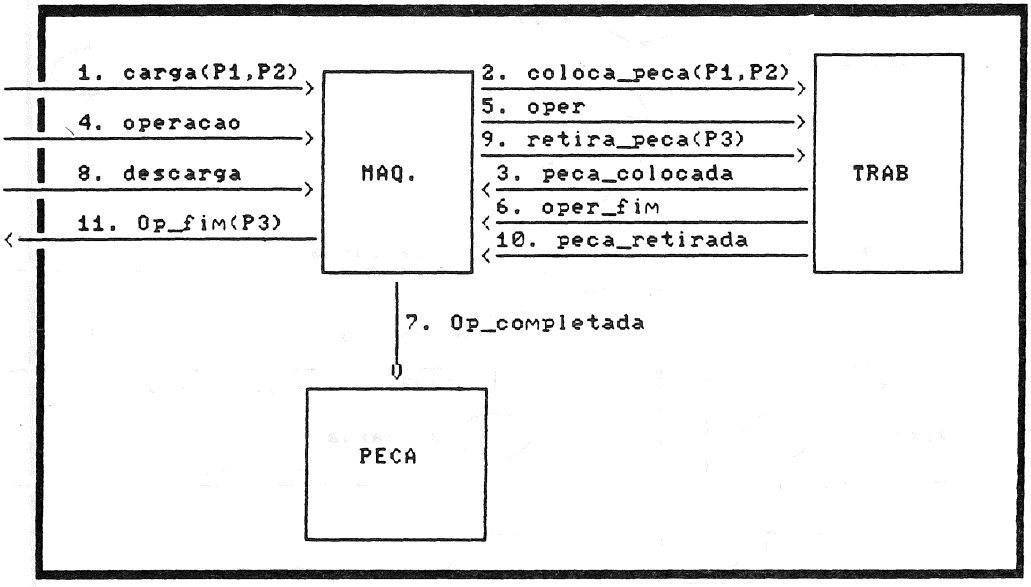


Fig. 5 Cenario de uma atividade

CONCLUSÕES

O artigo apresenta uma proposta para a modelagem de sistemas de manufatura que tem como base a metodologia orientada a objetos e o formalismo das redes de Petri. A modelagem orientada a objetos é adequada para a representação das entidades físicas de manufatura, permitindo tambem a representação das operações destas entidades. O formalismo das redes de Petri nos permite modelar aspectos de sincronização e paralelismo entre atividades e tambem serve como base para simulação do sistema de manufatura modelado.

BIBLIOGRAFIA

[ADI89] ADIGA, S. Software Modelling of Manufacturing Systems: A casefor an Object-Oriented Programming Approach. ANNALS OF OPERATIONS RESEARCH, 17(363-378), 1989.

[BAS89] BASTIDE, R. & SIBERTIN-BLANC, C. Conception Par Objets de Systemes Paralleles. In. SECOND INT. WORKSHOP ON SOFTWARE ENGINEERING AND ITS APPLICATIONS, 4-8 decembre 1989, Toulouse.

[BES85] BESSANT, J. The integration barrier; problems in the implementationof advanced manufacturing technology. Robotica. April-June 1985, p.97-103.

[BOO86] BOOCH, G. Object-Oriented Development. IEEE Transactions on Software Engineering , 12(2), Feb. 1986, p. 211-221.

[BRU86] BRUNO, G. & BALSAMO, A. Petri-net Based Object-Oriented Modelling of Distributed Systems. Proceedings. OOPSLA'86, Sep. 1986.

[COX86] COX, B.J. Object-Oriented Programming . Addison-Wesley. Reading MA, 1986.

[ENG85] ENGELKE, H. et alli. Integrated Manufacturing modeling system. IBM Journal of Res. Development. July 1985, p.343-355.

[GEN81] GENRICH, H.J. & LAUTENBACH, K. System Modelling with High-Level Petri Nets. Theoretical Computer Science, 13, 1981, p.109-136.

[GLA89] GLASSEY, R.C. & ADIGA, S. Conceptual Design of a Software Object Library for Simulation of Semiconductor Manufacturing Systems. Journal of Object-Oriented Programming, SIGS, New York, Nov/Dec 1989, p.39-43.

[*HEU88*] HEUSER, C.A. Modelagem de Sistemas com Redes de Petri. CPGCC, Porto Alegre, Maio, 1988.

[*MIY89*] MIYAGI, P.E. MFG/PFS Uma metodologia para o desenvolvimento do Controle de Sistemas Automatizados de Manufatura. In: 1. ENCONTRO REGIONAL DE AUTOMAÇÃO E INSTRUMENTAÇÃO. E S, 1989.

[*TAM84*] TAMM, B.; KITTNER, R. & VILIPYLD, J. A system for CAD/CAM software development and implementation. Computers in Industry . 5 (1984) , p.367-376.

[*TEL84*] TELICHEVESKY, R.G. Prospec: Um Formalismo para a Especificação de Sistemas de Fabricação. Dissertação de Mestrado, Porto Alegre, Junho 1984.

[*WAL89*] WALTER, C. Representação de Sistemas de Manufatura. Notas de Aula. CPGCC. Porto Alegre, RS. 1989.

SIMplex PARalelo (SIMPAR): FASE I

Alfonso Reinoza*, Marianela Lentini**, Alejandro Teruel*,
Lidia Da Silva¥, Alfredo Guillén¥,Tito Lozada§, Oscar Naím*

RESUMEN

El proyecto SIMPAR tiene como objetivo desarrollar una implementación paralela, en una red de transputadores, del método SIMPLEX para resolver problemas de programación lineal aplicados a la Industria Petrolera.

Este es un proyecto de investigación y desarrollo enmarcado en un esfuerzo conjunto entre MARAVEN, filial de Petróleos de Venezuela, y la Universidad Simón Bolívar.

En este trabajo presentamos una descripción de los objetivos y logros alcanzados en la primera fase del proyecto.

1. - INTRODUCCION

Los estudios realizados por la Gerencia de Tecnología de MARAVEN, a mediados del año 1988, mostraron la importancia y potencialidad del procesamiento paralelo. De hecho existían para la fecha más de 46 empresas fabricantes de equipos con paralelismo, habiendo desempeñado la industria petrolera internacional un rol protagónico en el aprovechamiento de este tipo de tecnologías. Urgía analizar en mayor detalle:

- El estado del arte de la tecnología, tanto en lo relativo a hardware como a software;
- La relación costo-beneficio de su uso;
- La viabilidad de desarrollar aplicaciones de envergadura con las herramientas disponibles,

* Departamento de Computación y Tecnología de la Información, Universidad Simón Bolívar
** Departamento de Matemáticas, Universidad Simón Bolívar
¥ I.M.G.
§ MARAVEN S.A.

- La capacidad nacional para asimilar y aplicar a corto plazo esta tecnología.

A tal fin se eligió como estrategia desarrollar un proyecto piloto que permitiese analizar estos factores. El proyecto constaría de una aplicación cuyo desarrollo exitoso mostrase la posibilidad de producir un impacto notorio que facilitase la difusión del procesamiento paralelo en la industria.

El Modelo del Area como se denomina al modelo de la planificación mensual de la producción de la refinería de Cardón, fue la aplicación seleccionada. Este presenta de manera evidente las características de un problema apropiado para el paralelismo: volumen considerable de datos, cálculos masivos y exigencias del usuario en cuanto al tiempo de respuesta.

Este modelo es un problema de programación lineal generado vía el RPMS ("Refinery Petrochemical Modeling System") de Bonner y Moore y resuelto con el MPSX (Mathematical Programing System Extended) de IBM. La operación del modelo, desde la captación de datos hasta el análisis de resultados, se hace en un ambiente de mainframe (IBM 3090) en la modalidad "batch", creando dificultades a las altas exigencias interactivas que este tipo de problema requiere.

En enero de 1989, se inició el análisis para formular un proyecto de paralelización del Modelo del Area en arquitecturas de procesadores paralelos con memorias locales (transputadores). La esencia de este proyecto consiste en desarrollar software paralelo para resolver problemas de programación lineal i.e. el Método Simplex. La poca familiaridad con la tecnología de los transputadores planteó la necesidad de desarrollar un prototipo rápido con los siguientes objetivos:

- Introducir la tecnología a la industria, en lo relativo a:
  - Hardware,
  - Metodologías de desarrollo,
  - Herramientas y métodos de productividad.
- Validar la hipótesis de paralelización del Método Simplex para resolver el Modelo del Area.

- Utilizarlo como base para la construcción de un software de programación lineal, con la calidad de un producto de mercado.

Para lograr satisfacer estos objetivos se requerían recursos humanos especializados en las siguientes áreas:

- Paralelismo
- Programación Lineal
- Análisis Numérico

El equipo inicial lo conformaron R. Karia en Paralelismo, A. Reinoza en Programación Lineal, M. Lentini en Análisis Numérico y T. Lozada como implementador. Una revisión de la literatura en el área, mostró que el proyecto SIMPAR constituía un desarrollo pionero, que requeriría de investigación para formular un modelo conceptual del Método Simplex en paralelo y resolver aspectos como:

- La paralelización de los procesos numéricos;
- La organización de los datos del problema en las memorias locales;
- La arquitectura paralela más apropiada a la aplicación;
- Los algoritmos de comunicación entre los transputadores.

La complejidad de estos aspectos y la necesidad de tener respuestas razonablemente rápidas a las interrogantes planteadas, condujo a una fase preliminar de desarrollo de un prototipo exploratorio según la clasificación de Riddle and Williams [7]. Para ello, se seleccionó la versión más sencilla del Método Simplex (Dantzig 1949) [2] con potencial para resolver el Modelo del Area simplificándose la plataforma de desarrollo a:

- Una arquitectura de cuatro transputadores con posibilidades muy simples de comunicación;
- El lenguaje C paralelo (versión 3L) en lugar del OCCAM (lenguaje natural para transputadores).

En marzo de 1990 se dispuso de un prototipo exploratorio, que validó la hipótesis sobre el potencial de paralelización del Método Simplex. De hecho, el tiempo de ejecución ("elapsed time") de algunos problemas de programación lineal sobre el prototipo arrojó datos comparables al tiempo de ejecución sobre MPSX en el IBM 3090.

El próximo paso consiste en desarrollar un software de calidad de producción. Dado el carácter de proyecto de investigación y la insuficiencia de recursos humanos propios, la Gerencia de Tecnología de Maraven decidió plantear el proyecto como un punto inicial de enlace en el área de informática, entre MARAVEN y la Universidad Simón Bolívar.

Esta idea cristalizó y hoy, bajo el nombre de SIMPAR, se está ejecutando un proyecto conjunto, organizado en tres fases. La primera fase consistió en una evaluacion detallada del prototipo y elaboración de un diseño basico, especialmente en cuanto a su potencial de escalabilidad a un número mayor de transputadores y la posibilidad de reutilizar porciones del diseño o el código (o ambos) del prototipo en el desarrollo del software piloto.

En este articulo presentamos la evaluación del prototipo en la sección 2, donde se consideran aspectos paralelos, numéricos y de la ingeniería de la programación. En la sección 3 se dan algunas conclusiones y recomendaciones.

2.- EVALUACION DEL PROTOTIPO

El prototipo de SIMPAR fue evaluado a la luz de sus metas y con los criterios y tecnicas de las areas del conocimiento sobre las que se apoya

- Paralelismo,
- Analisis Numerico,
- Programacion lineal,
- Ingenieria de la programación,

Brevemente comentaremos los resultados.

2.1.- PARALELISMO

Con respecto al **paralelismo**, se evaluaron los siguientes aspectos:

- Diseño conceptual de la versión paralela del Método Simplex;
- Comunicaciones;
- Métricas de Desempeño.

Un análisis detallado de estos aspectos se encuentra en el reporte interno [5]. No obstante quisieramos resaltar que la concepcion paralela del método ha sido afortunada, destacando:

- La distribucion por columnas de la matriz de coeficientes en las memorias de los transputadores;
- La replicacion del método Simplex en cada transputador;
- La granularidad del paralelismo.

Sin embargo la simplicidad del diseño del prototipo lo ata fuertemente a la arquitectura de cuatro procesadores en una relación centralizada (master/slave). Haber adoptado esta arquitectura y la técnica de comunicación sencilla pero limitada, ha sido un factor contrario a las posibilidades incrementales del prototipo. La figura 1 presenta los conceptos aquí planteados,

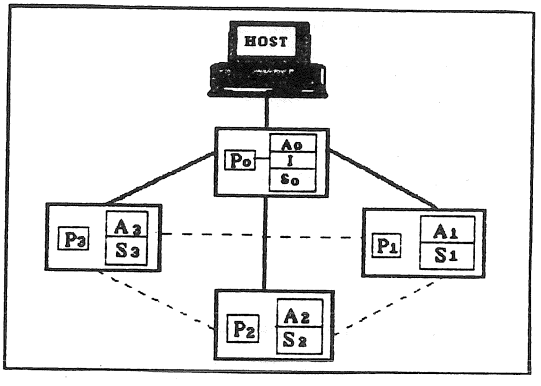


Figura 1

donde P_j denota el j-ésimo procesador, A_j el j-ésimo bloque de columnas de la matriz A, S_j la j-ésima replica del algoritmo Simplex e I representa el algoritmo coordinador de las comunicaciones.

En lo relativo a su desempeño, el prototipo fue evaluado con las siguientes métricas:

- Aceleramiento ("speedup");
- Eficiencia;

- Fracción Serial Empírica:
- Descomposicion del tiempo de ejecucion por rutina.

Dichas métricas fueron aplicadas para hallar respuestas a las siguientes preguntas:

- ¿Es aceptable el rendimiento del prototipo?
- ¿Cual sera el rendimiento del sistema con 16 transputadores?
- ¿Existen cuellos de botella o rutinas cuya mejora influyen críticamente en el rendimiento del prototipo?

2.1.1 Aceleramiento

El aceleramiento es la medida de cuánto más rápido se logra correr un algoritmo sobre N procesadores en comparación con su tiempo de ejecución en un procesador.

Idealmente uno aspira a que el aceleramiento de un algoritmo paralelo sea lineal, es decir que si se tienen N procesadores, se lograra ejecutar el algoritmo N veces más rápido que si se dispone de un solo procesador.

En la práctica el aceleramiento rara vez es lineal, por una serie de factores como:

- Tiempo adicional que se necesita para sincronizar los N procesadores o para que éstos se comuniquen entre ellos,
- El algoritmo tiene partes que deben ejecutarse serialmente o en los cuales no hay suficiente trabajo disponible para mantener ocupados los N procesadores.

En el caso del prototipo de SIMPAR, sobre 4 procesadores se obtiene un aceleramiento que se acerca a 3.5 a medida que crece el problema, en la figura 2 se muestra este comportamiento. Este es un buen resultado para un sistema de cuatro procesadores, pero ¿qué nos dice respecto al aceleramiento potencial en un sistema de dieciseis transputadores? Para hallar la respuesta a esta pregunta debe estudiarse la próxima métrica.

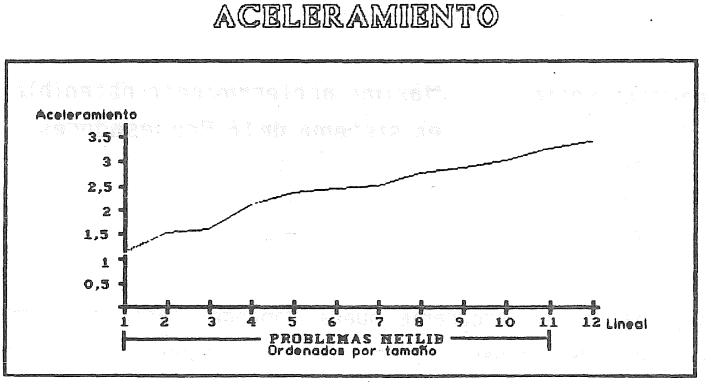


Figura 2

2.1.2 Fracción Serial Empírica

La fracción serial del programa es aquella fracción del tiempo de ejecución del programa en que la ejecucion es estrictamente serial, es decir en que el sistema se comporta como si un solo procesador estuviese ejecutando.

La fracción serial es crítica para el rendimiento de cualquier programa paralelo. En la figura 3 se ilustra la relación entre fracción serial y aceleramiento.

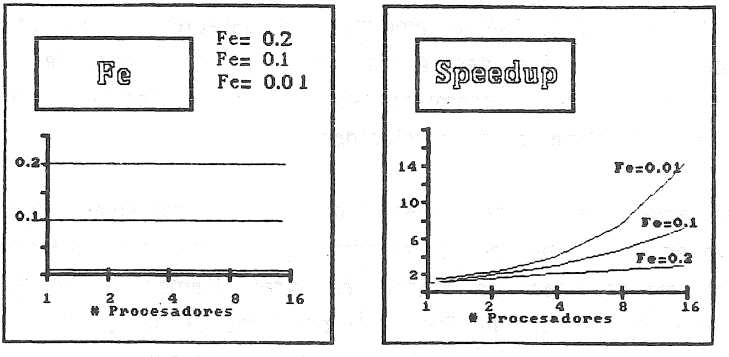


Figura 3

Se observa, como lo muestra la tabla siguiente, un incremento en el aceleramiento al disminuir la fracción serial:

Fracción serial	Máximo aceleramiento obtenible en sistema de 16 Procesadores
20%	4.0
10%	6.4
1%	13.9

La fracción serial de un programa puede depender tanto del número de procesadores como del tamaño del problema. Al aumentar el número de procesadores puede aumentar el tiempo requerido para sincronizar los procesadores; esto incrementa la fracción serial del programa. Al aumentar el tamaño del problema, la fracción serial puede aumentar (señal de posibles cuellos de botella en el algoritmo o en los recursos disponibles), puede mantenerse constante o puede incluso disminuir. El último caso (la fracción serial disminuye) es el ideal.

Dado que es sumamente difícil determinar la fracción serial teórica de un programa, se recurre a la fracción serial empírica, la cual es una aproximación experimental.

Los once problemas tomados de **Netlib** (biblioteca estándar de pruebas para cualquier paquete de Programación Lineal) muestran que el prototipo de SIMPAR tiene una fracción serial empírica que pareciera estabilizarse en 20%. Este dato por si solo es desalentador ya que de ser cierto, recorta las máximas aspiraciones de aceleramiento sobre 16 transputadores a una poco satisfactoria cuadruplicación de la velocidad.

Dado que el tamaño máximo de la matriz de los once problemas tomados de **Netlib** es apenas del orden de 10^3 elementos, se hizo una prueba adicional con un problema recortado del Modelo del Area (llamado Lineal en este reporte) cien veces más grande. Para este problema, se obtuvo una fracción serial empírica de 6%, lo cual permite proyectar un aceleramiento de 8.42 en un sistema de 16 procesadores.

¿Qué significa la discrepancia entre los dos resultados? La discrepancia es un resultado alentador ya que indica la posibilidad que la fracción serial del programa decrezca con el tamaño del problema. De resultar cierto no es descabellado pensar en la posibilidad que el Modelo del Area alcance un aceleramiento de 14 sobre un sistema de 16 transputadores.

Sin embargo conviene alertar que existen varios factores por los cuales la aseveración anterior sólo puede hacerse de manera tentativa. En particular es necesario realizar más pruebas con otros programas grandes para confirmar el resultado obtenido. Adicionalmente deben hacerse pruebas con más procesadores, ya que con los datos obtenidos es imposible usar la fracción serial empírica para estudiar posibles cuellos de botella.

En resumen, las pruebas indican que el aceleramiento obtenible sobre un sistema de 16 procesadores pueden variar desde 4 (estimado pesimista) hasta 14 (estimado optimista) con una alta probabilidad de que se obtenga un aceleramiento entre 6 y 12, para el algoritmo actual.

También se apunta hacia la necesidad de realizar pruebas adicionales sobre un prototipo sencillo de 8 y de 16 transputadores.

2.1.3 Descomposición del tiempo de ejecución por rutina

Dado que se carece de datos suficientes para aplicar la métrica anterior para detectar posibles cuellos de botella, se apeló a un estudio del consumo de tiempo de ejecución por rutina del prototipo.

Fundamentalmente hay cinco procesos ejecutándose en el prototipo de SIMPAR, un proceso maestro llamado INTERFAC que se encarga de sincronizar y coordinar a cuatro procesos esclavos llamados $SIMPLEX_i$.

El estudio de los procesos esclavos muestra una buena distribución de la carga de trabajo; casi no hay momentos en que un proceso esclavo quede ocioso esperando a que otro termine su trabajo. Adicionalmente se muestra que una de las rutinas, PIVOTEO, consume el grueso del trabajo; para ser precisos, más del 86% del tiempo de ejecución de un proceso esclavo

$SIMPLEX_1$ se lo lleva PIVOTEO, ver figura 4. Esta es una excelente noticia, ya que PIVOTEO es una rutina de cálculo que no requiere de sincronización o comunicación alguna durante su ejecución. Adicionalmente cada PIVOTEO del proceso esclavo trabaja con una partición diferente de datos, por lo que el paralelismo que se está obteniendo en PIVOTEO es muy cercano al óptimo. Ello reivindica la estrategia de paralelización, en particular la técnica de "pricing" Múltiple [6].

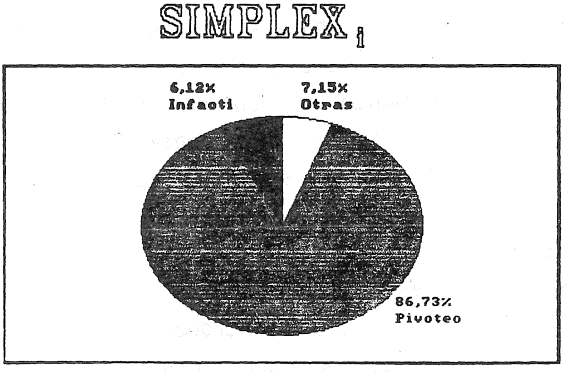


Figura 4

El estudio del proceso maestro INTERFAC muestra también una rutina, REC_SWT que consume la mayoría abrumadora del tiempo. Un análisis detallado de la misma, muestra que el tiempo mayoritario de REC_SWT corresponde a la espera por la terminación de PIVOTEO por parte de los procesos esclavos. La figura 5 ilustra esta situación.

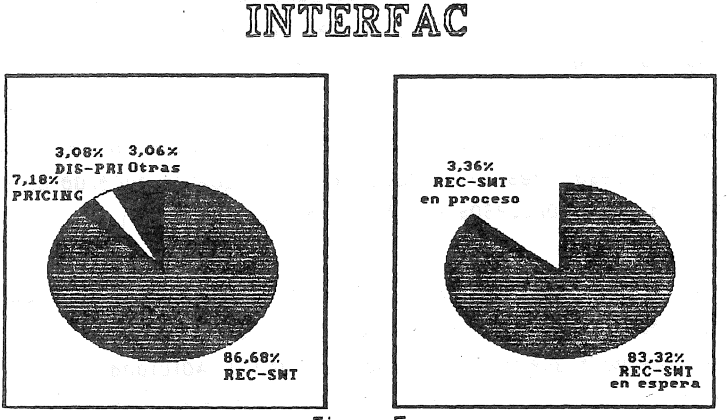


Figura 5

Todo ello indica tentativamente que los algoritmos seleccionados son los apropiados y que no existen cuellos de botella apreciables. Estos resultados por supuesto están sujetos a cambios una vez que se incorpore la funcionalidad completa del sistema de Programación Lineal, ya que el prototipo de SIMPAR no maneja adecuadamente la acumulación de errores de redondeo. Estos aspectos son importantes; a título de ejemplo, conviene recordar que muchos sistemas comerciales se ven obligados a reinvertir la base cada cincuenta iteraciones para evitar problemas de estabilidad numérica. El proceso de reinversión es costoso, y probablemente se requerirá realizarla varias veces durante la ejecución del Modelo del Area.

Adicionalmente, los resultados indican la posibilidad que las comunicaciones no lleguen a sobrepasar el 14% del tiempo de ejecución del algoritmo, incluso para topologías muy sencillas como la del anillo. Dada la simplicidad del anillo y su fácil y cómoda escalabilidad, conviene desarrollar un prototipo de 16 transputadores en anillo para estudiar el impacto de sus requerimientos de comunicación. Si el impacto es bajo, los beneficios para el sistema final son muy altos, si el impacto es alto, habrá que descartar el anillo, pero se habrán obtenido resultados valiosos para estudiar qué configuración sería la más adecuada.

2.2.- ASPECTOS NUMERICOS Y HEURISTICOS

La evaluación de los algoritmos numéricos y las heurísticas usadas en el prototipo, así como los resultados de las pruebas con los problemas de **Netlib** y algunos ejemplos aportados por la Gerencia de Planificación de MARAVEN, indican que que hubo una selección afortunada de:

- La versión original de Dantzig (tableau completo) [2] del Método Simplex;
- La técnica heurística de "Pricing" Múltiple [6];
- Los criterios heurísticos de selección del pivote.

Sin embargo, aunque estos algoritmos fueron probados, aún persisten algunos problemas:

- Estabilidad numérica. Para algunos problemas de **Netlib** el prototipo arroja resultados incorrectos y en algunos casos no los puede resolver, ver [4].

- Ciclos. Este fenómeno, que ocurre cuando el problema de programación lineal es degenerado, i.e., con frecuencia los criterios de selección del pivote no pueden resolver ambiguedades y se generan repeticiones de las mismas bases, haciendo que el programa caiga en un ciclo infinito. Para evitar esta anomalía, hemos incorporado al nuevo diseño un procedimiento heurístico anti-ciclos basado en los trabajos realizados por el grupo del Systems Optimization Laboratory de la Universidad de Stanford [3].
- Robustez: Al variar el número de procesadores o el tamaño del "pricing set", el software falla con varios problemas de **Netlib**. Esto es una clara indicación de que puede haber problemas en la implementación de las heurísticas de "pricing" y otros problemas de implementación.
- Entonación de tolerancias: Es posible que algunos de los errores que persisten en el prototipo, sean originados por los parámetros que controlan la toma de decisiones. Estas tolerancias deben ser cuidadosamente ajustadas, una tarea delicada que requiere de pruebas exhaustivas y variadas.

2.3.- INGENIERIA DE PROGRAMACION

El prototipo de SIMPAR resultó ser más un prototipo exploratorio que uno incremental. Como tal cumple sus cometidos admirablemente; demostró la factibilidad de dominar una nueva tecnología, demostró la factibilidad de paralelizar SIMPAR y dió indicios de los resultados esperables al expandir su plataforma de hardware a dieciseis transputadores. Sin embargo no fue desarrollado bajo criterios de calidad de producto final. La cohesión de algunos de sus módulos es baja, llegando en algunos casos a cohesión coincidental, el acoplamiento entre algunas rutinas es excesivamente alto.
El prototipo de SIMPAR refleja un proceso de aprendizaje y correcciones locales que conducen a un producto frágil ("brittle") respecto a su modificabilidad.

Estos resultaoos nos condujeron a rediseñar totalmente el producto de "software". El rediseño de SIMPAR siguiendo criterios de calidad, puso de manifiesto una serie de problemas no contemplados en el prototipo y mejoró

considerablemente el diseño con respecto a criterios tradicionales como se muestra en la figura 6.

CRITERIOS DE DISEÑO

		PROTOTIPO	DISEÑO ACTUAL
ACOPLAMIENTO	# Máx de Parámetros	22	5
	# de Par. de Control	7	1
	Camino máximo que recorre un dato	5	4
COHESION	# máximo de funciones entrelazadas	8	3
	Tipo de Cohesión	Coincidental (Nivel 1)	Secuencial (Nivel 6)
ENCAPSULAMIENTO		NINGUNA	Tableau, PPL (En estudio) (Todas las variables globales)
# MODULOS		20	100

Figura 6

Queda pendiente la revisión de ciertos aspectos del nuevo diseño con respecto al paralelismo (por ejemplo, como encapsular la comunicación de manera de lograr cierta independencia respecto a la configuración de hardware sin por ello perder excesivamente eficiencia).

2.4.- LAS HERRAMIENTAS

Hemos tenido la oportunidad de evaluar las herramientas de desarrollo de software para los transputadores y en general hemos encontrado que su estado es mucho más rudimentario y volátil que sus equivalentes sobre máquinas paralelas. En particular, las herramientas de apoyo a la depuración de programas son sumamente limitante, pese a los nuevos problemas que surgen en programación paralela, tales como abrazos mortales ("deadlocks")

2.5.- ANALISIS DE LA PLATAFORMA DE HARDWARE

Durante la fase 1 del proyecto se analizaron en detalle dos problemas importantes:

- Los requerimientos de memoria del modelo de área,
- La topología de interconexión entre transputadores.

Ambos problemas tienen repercusiones críticas sobre la plataforma de hardware existente. Dicha plataforma consta de:

- Dos tarjetas Quadputer, para un total de ocho transputadores con 4 Megabytes de memoria cada uno;
- Ocho transputadores con 2 Megabytes de memoria cada uno conectados en tarjetas B008 de INMOS.

Un sistema paralelo es más facil de manejar cuanto más homógeneo sea. La configuración actual rompe con la homogeneidad en tres puntos:

- Se manejan dos tecnologías de tarjetas (Microway e Inmos),
- Los transputadores tienen diferentes recursos de memoria (4Mb,2Mb);
- La red de interconexion entre transputadores es heterogenea (una conexión fija en forma de grafo completo de 4 nodos para los Quadputers y un switche reconfigurable para las B008).

Esta heterogeneidad tiene las siguientes implicaciones, de conservarse la configuración actual:

- Para manejar los requerimientos del modelo de área (unos 38 Megabytes de memoria como mínimo), es necesario implementar algún mecanismo de manejo de memoria que o bien rompa con la distribución equitativa de carga de trabajo existente actualmente o bien implemente memoria virtual (bajo esquema de cache o de disco o ambas).
- No es posible conectar los transputadores en una topología de hipercubo ni tampoco en una malla cuadrada de 4x4 transputadores.
- Debe buscarse una topología de conexión adecuada. No se plantearán aquí todas las alternativas que fueron consideradas, ellas están detalladas en la referencia [5].

Dadas las siguientes consideraciones,

- No existe un manejador de memoria virtual confiable a nivel comercial para transputadores;
- El tiempo de desarrollo de tal manejador es no trivial;
- Hacer "swapping" desde el disco del PC anfitrión degrada el sistema de manera totalmente inaceptable (estimado del orden de 24 horas adicionales de tiempo de ejecución para el Modelo del Area);
- El uso de los 4 Megabytes por transputador como "cache" degrada el sistema además de romper con la homogeneidad de distribución de carga;
- Hay indicios que las comunicaciones no son críticas y que por ende no hace falta una topología tan sofisticada como el hipercubo y puede ser que la topología de anillo baste;
- El hardware actual puede armarse en forma de anillo.

se recomienda fuertemente determinar a través del estudio de un prototipo de 16 transputadores conectados en anillo utilizando 2 Megabytes de memoria por transputador, si la topología puede limitarse a un anillo.

Dependiendo de los resultados del prototipo del anillo, existen varias opciones a tomar sobre la plataforma futura para el proyecto, que se detallan en la referencia [5].

2.6.- ASPECTOS METODOLOGICOS

Conviene insistir en que el proyecto no sólo maneja una tecnología del estado del arte (paralelismo) sino que también busca controlar la calidad del producto final así como del proceso de desarrollo mediante la aplicación de técnicas de Ingeniería de la Programación.

De esta manera, se trabaja con un modelo de desarrollo de cascada enriquecido con las siguientes nociones:

- Prototipos exploratorios e incrementales;
- Manejo del concepto de calidad;
- Actividades no limitadas al diseño de software, sino también al diseño más apropiado de la configuración de hardware;

- Inspecciones informales de tangibles;
- Manejo (sencillo) de aspectos de riesgos e incertidumbre, por ejemplo mediante la adaptación del modelo COCOMO intermedio de estimación de tiempos, ver [1].

A efecto del modelo COCOMO y a título de ejemplo, considérese el impacto de distribuir la carga heterogéneamente entre los transputadores con el objeto de cumplir con la configuración peculiar de memoria existente en la plataforma de hardware actual. Aplicando el modelo COCOMO se puede predecir que dicha distribución obligará a invertir entre 11 y 30% de trabajo adicional en el proyecto.

También es conveniente recalcar que se aplica una metodología de programación en paralelo sencilla, desarrollada en los cursos del área de Paralelismo de la Maestría en Ciencias de la Computación de la Universidad Simón Bolívar, que ha ayudado a evaluar el prototipo y sus necesidades de coordinación, identificando rápidamente potenciales cuellos de botella y concentrando los esfuerzos de análisis en esos puntos.

3.- CONCLUSIONES

De los resultados obtenidos en esta fase del proyecto, podemos concluir que el prototipo cumplió exitosamente con sus objetivos. Hoy podemos afirmar que las interrogantes planteadas en cuanto a la posibilidad de resolver el Modelo del Area de Maraven S.A., con un tiempo de respuesta aceptable, han quedado despejadas.

Los resultados en el área de paralelismo muestran con altas probabilidades un aceleramiento entre 8 y 12 sobre 16 procesadores. Además se vislumbra que una configuración de anillo sea suficiente, sin embargo, para confirmarlo se recomiendan más pruebas con el prototipo adaptado a 16 transputadores.

Se evidenció además, que la versión paralela del método Simplex implementada en el prototipo, ha sido una buena selección; no hay síntomas de cuellos de botella, la fracción serial es baja y muestra una tendencia a decrecer al aumentar el tamaño del problema.

Con respecto al análisis de los algoritmos numéricos y heurísticos, es preciso introducir procedimientos de control de errores numéricos, afinar los parámetros de control y enriquecer las heurísticas para la selección de pivotes y control de ciclos.

Sobre los aspectos metodológicos y de ingeniería de la programación, se puede concluir que se ha logrado un buen diseño (bajo acoplamiento, alta cohesión y modularidad) de la versión serial de SIMPAR. Para la versión paralela es necesario definir una notación que refleje en forma clara y precisa el paralelismo y la comunicación entre módulos.

En líneas generales, la Fase I ha mostrado la viabilidad para desarrollar un software de producción

Al margen de los aspectos técnicos, ha habido logros académicos que ameritan mención. Alrededor del proyecto se ha generado una creciente actividad de investigación que se ha manifestado en varias tesis de post-grado, entre ellas:

- Un generador de métricas de desempeño
- Apoyo a la generación de casos de prueba
- Algoritmos de programación lineal en paralelo:
  - Otras versiones del método Simplex
  - Algoritmos de puntos interiores
- Una versión paralela del algoritmo de Lemke con variables acotadas.

Conviene además mencionar que tesis de maestrías han fortalecido al proyecto SIMPAR, entre otras:

- Algoritmos de Comunicación de Hipercubos Generalizados.
- Descomposición de Dominios (Conjuntamente con INTEVEP, S.A.)

Como conclusión final, cabe destacar que los resultados de esta primera fase han sido altamente satisfactorios para ambas partes, tanto desde el punto de

vista técnico como de la visión de acercamiento entre las dos instituciones involucradas, MARAVEN y U.S.B.

4.- REFERENCIAS

1 BOEHM W., Barry: Software Engineering Economics. Prentice Hall (1981)

2 DANTZIG, George B.: Linear Programming and Extensions
Princeton University Press (1963).

3 GILL, Philip E. y MURRAY, Walter y SAUNDERS, Michael A. y WRIGHT, Margaret H.: A Practical Anti-Cycling Procedure for Linear and Nonlinear Programming. Technical Report SOL 88-4 Stanford (July 1988)

4 GUILLEN, Alfredo y LENTINI, Marianela: Consideraciones Numéricas en SIMPAR (Fase I). Reporte técnico Nº IT-1991-002. Proyecto conjunto USB-MARAVEN, S.A. Departamento de Computación y Tecnología de la Información. Universidad Simón Bolívar (1991).

5 NAIM, Oscar y TERUEL, Alejandro: Consideraciones sobre el Paralelismo en SIMPAR (Fase 1). Reporte técnico interno Nº IT-1991-001.
Proyecto conjunto USB-MARAVEN. Departamento de Computación y Tecnología de la Información. Universidad Simón Bolívar (1991).

6 ORCHARD-HAYS, William: Advanced Linear-Programming Computing Techniques. McGraw-Hill (1968).

7 RIDDLE, W. y WILLIAMS, L.G.: Software Environments Workshop Report. ACM SIGSOFT Software Engineering Notes, Vol. II, No. 1, p.p. 73-102, (1986).

ADQUIRINDO VANTAGEM COMPETITIVA ATRAVÉS DE RECURSOS DE INFORMAÇÃO

Luiz Henrique Boff, Ricardo Teitelroit e Érica Rosalba Hallmann Duarte
Mestrandos do Programa de Pós-graduação em Administração
Área de Produção e Sistemas
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Av. João Pessoa, 52 - 1º. andar - Sala 11
90040 Porto Alegre(RS) - BRASIL
Telefone: (0512)25-6533 - Fax: (0512)25-5253

RESUMO

Este trabalho visa fornecer uma visão geral do significado de vantagem competitiva e sua relação com os Recursos de Informação. São definidos seus conceitos básicos, sua importância como fator estratégico para as organizações e apresentados modelos de identificação de oportunidades e fatores que afetam a aplicação de Sistemas de Informação para a obtenção de vantagem competitiva.

ABSTRACT

This article seeks to give a broad view of the meaning of competitive advantage and the link with Information Resources (IR). Basic concepts are defined, its relevance as an strategic factor to the organizations, and it shows some models to identify opportunities created by IR to gain competitive advantages.

INTRODUÇÃO

A relação entre competitividade e Sistemas de Informação (SI) tem sido evidenciada por diversos autores, principalmente a partir da segunda medade da década de 80.

Os estudos de Brancheau e Wetherbe (BRANCHEAU, 1987) apontam um vertiginoso crescimento da vantagem competitiva como aspecto chave das preocupações dos administradores de SI, relacionando-os diretamente com as questões estratégicas das organizações.

Porter e Millar (PORTER, 1985) enfatizam o significado estratégico que os Recursos de Informação (RI) possuem na transformação e na natureza dos produtos, processos, organizações, indústrias e até da própria competição.

Na medida em que as organizações buscam uma maior adaptação aos novos desafios da economia mundial, cresce a necessidade de desenvolvimento científico e tecnológico e de utilização de recursos para obter vantagem competitiva.

Este trabalho busca contextualizar o assunto quanto à sua definição e aplicação nas organizações. A primeira parte apresenta o conceito de vantagem competitiva e sua relação com os RI. Na segunda, são abordados modelos para detecção de oportunidades no uso de RI para obter vantagem competitiva. Na terceira parte, são relatados os fatores inibidores e facilitadores no uso de RI no contexto organizacional. Finalmente, a conclusão é uma visão dos desafios a serem enfrentados pelas organizações diante do ambiente que se configura.

1. VANTAGEM COMPETITIVA E RECURSOS DE INFORMAÇÃO

Cada vez mais, com a introdução de novas tecnologias dentro das fábricas, os RI têm afetado os processos nas organizações e os próprios produtos que são criados. São transformações que aparecem em toda a cadeia, desde a concepção até a efetiva utilização de uma mercadoria

pelo consumidor. Ao longo das atividades envolvidas, estão disponíveis recursos técnicos que geram e transferem informação, e esta, por sua vez, serve aos processos e agrega conteúdo aos produtos.

1.1. Sistemas de Valor e Recursos de Informação

Conforme Porter e Millar (PORTER, 1985), o papel dos RI na competição pode ser entendido pela análise do que conceituam por uma ***cadeia de valor***. Essa cadeia é definida como sendo o trabalho que uma organização executa em seu negócio, dividido em atividades econômicas e tecnológicas, que são chamadas de ***atividades de valor***. O valor que uma empresa cria é determinado pelo que o consumidor deseja pagar pelo seu produto ou serviço. Para obter vantagem competitiva, uma organização precisa executar tais atividades buscando reduzir os custos de produção e efetivá-las de forma que elas conduzam a uma diferenciação.

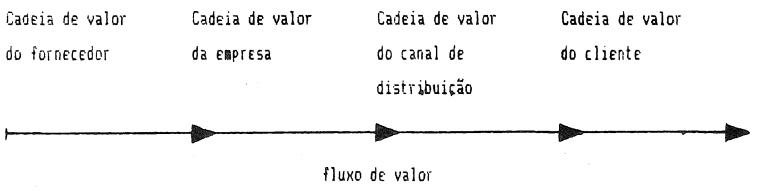
As atividades de valor agrupam-se em primárias e de suporte. As atividades primárias envolvem a criação propriamente dita dos produtos, marketing, distribuição e suporte e serviços de pós-venda. Atividades de suporte fornecem a infra-estrutura e os recursos que permitem às atividades primárias serem executadas.

A cadeia de valor de uma empresa é um sistema de atividades interdependentes, interligadas de maneira que a forma de execução de uma atividade afeta o custo e/ou a eficácia de outras. Além de produzirem *trade-offs* (perdas na execução de uma atividade em função do favorecimento da de outras) que precisam ser otimizados, as ligações requerem uma coordenação entre as atividades. Uma administração cuidadosa dessas ligações é, muitas vezes, uma importante fonte de vantagem competitiva, que acaba fazendo parte da estratégia da organização.

A cadeia de valor de uma determinada empresa está embutida em um ambiente maior chamado ***sistema de valor***. As ligações entre atividades de valor não se restringem à

cadeia de valor interna de uma organização, interconectando-se, também, com as cadeias de valor dos fornecedores, dos canais de distribuição e dos clientes (figura 1). Dessa forma, há uma necessidade de ampliar os esforços de otimização e coordenação a essas ligações externas.

FIGURA 1.1 - SISTEMA DE VALOR



Fonte: Porter, 1985

Na busca de vantagem competitiva as organizações diferem quanto ao escopo competitivo, a partir de quatro dimensões: segmentação de mercado (características do público consumidor que define o destino dos produtos), grau de integração vertical (maior ou menor independência de produção), abrangência geográfica (área onde a empresa atua e distribui seus produtos) e ramo de atividade (ambiente industrial em que a empresa compete).

Os RI atuam em todos os pontos de uma cadeia de valor, transformando a forma como as atividades de valor são executadas e a natureza das ligações entre elas. Geralmente, também é afetado o escopo competitivo e remoldada a forma como os produtos atingem as necessidades dos consumidores.

Toda atividade de valor possui componentes físicos - tarefas requeridas para a execução de atividades - e componentes de processamento de informação - captura, manipulação e distribuição de dados envolvidos na execução das atividades.

Os recursos de informação geram maior número de dados à medida que a organização efetua suas atividades. Além disso, permitem coletar informações que não estavam disponíveis anteriormente. Esses recursos aumentam a

capacidade de análise e o volume de variáveis que uma organização pode considerar ou controlar. Dessa forma, permite às empresas uma maior habilidade no trato das ligações entre as atividades de valor dentro e fora do seu ambiente de produção.

Os efeitos desses recursos têm um alcance significativo, tanto que estão causando um novo problema aos executivos: a existência de uma quantidade excessiva de informação.

1.2. Tecnologia da Informação X Informação

Muitos estudos tratam sobre SI relacionados com vantagem competitiva. Entretanto, a Tecnologia da Informação é geralmente abordada de forma exclusiva (CLEMONS, 1986; OLIVEIRA, 1987; BAKOPOULOS, 1985a; MENDES, 1987; BENJAMIN, 1984), não havendo uma distinção entre dois elementos distintos: a Informação e a Tecnologia da Informação. Segundo King, Grovel e Hufnagel (KING, 1989), existe a necessidade de fazer uma separação entre esses conceitos, pois a aplicação estratégica dos RI pode ser feita de forma diferente.

Os autores se baseiam em Bakopoulus (BAKOPOULOS, 1985b) para conceituar **Tecnologia da Informação** como *um conjunto de recursos não-humanos dedicados ao armazenamento, processamento e comunicação de informação, e o modo como esses recursos são organizados em um sistema capaz de efetuar um conjunto de tarefas*. A tecnologia inclui recursos de hardware e software.

Informação é definida por King, Grover e Hufnagel (KING, 1989) como *dados que foram analisados de tal forma que alteram as expectativas ou a visão sobre alternativas que estão disponíveis*. Há também uma referência ao software analítico que facilita o uso dos dados.

A Tecnologia da Informação adiciona valor através dos dispositivos utilizados no processo de manuseio da Informação. O potencial de adição de valor é considerado baixo, pois deprecia com o tempo, principalmente com o

surgimento de novas tecnologias.

O valor adicionado pela Informação se refere a como os depósitos de dados colecionados pelos sistemas existentes podem ter novos usos. O dado, considerado como matéria-prima da informação, pode ser usado de diversas formas, e o foco no conteúdo pode sugerir oportunidades não evidenciadas pela tecnologia.

Ao contrário da Tecnologia da Informação, a Informação não deprecia seu valor. A partir do momento em que é utilizada ela fica mais valiosa. Quanto mais são atingidos usos potenciais e seu significado é gerado e melhor percebido pelo homem, mais valor é produzido.

Conforme o modelo conceitual utilizado por Fensterseifer (FENSTERSEIFER, 1990), desenvolvido para avaliar o impacto potencial das tecnologias flexíveis na competitividade das empresas, o valor de uma organização possui dois componentes: o valor presente das *atividades atuais* e o valor presente das receitas esperadas na realização de *novas atividades*.

O valor derivado das *novas atividades* depende diretamente da natureza das *atividades atuais*, podendo-se imaginar um crescente de aprendizado conforme patamares tecnológicos atingidos. Cada nova atividade traz consigo a base para a ascenção a outro patamar. Isso é feito através da otimização das atividades atuais e do preparo para atividades futuras.

Podemos dizer que a aplicação de Tecnologia da Informação, normalmente desenvolvida tendo as atividades atuais como referencial, permite maior precisão na análise de investimentos do que a Informação. Esta assemelha-se a investimentos em pesquisa e desenvolvimento (P&D), no que diz respeito à incerteza na mensuração de seu potencial técnico e comercial. Seu valor só se concretiza pela utilização derivada de sua existência. É preciso, portanto, uma cuidadosa seleção de ferramentas de avaliação, principalmente dos benefícios possíveis.

A falta de distinção entre os RI pode ser a causa

de abordagens práticas equivocadas de desenvolvimento e aplicação estratégica de SI. As idéias de utilização de Tecnologia da Informação aparecem mais facilmente e anteriormente às da Informação, podendo implicar que estas sejam colocadas em segundo plano, mesmo que possam gerar potencialmente mais benefícios.

2. MODELOS PARA DETECÇÃO DE OPORTUNIDADES NO USO DE RECURSOS DE INFORMAÇÃO

Num contexto organizacional, pode-se observar três componentes básicos da estratégia corporativa: (a) estratégia interna, que diz respeito ao desenvolvimento da eficiência e eficácia da estrutura organizacional; (b) estratégia competitiva, que enfoca o comportamento da organização dentro de seu ramo de negócio; e (c) estratégia de portfólio de negócio, que se refere à escolha de qual indústria competir e como posicionar a organização nessa indústria.

2.1. Recursos de Informação e Estratégia Interna

O ambiente interno das organizações tem sido classificado como o foco tradicional no uso de sistemas de informação, visando o aumento da eficiência e eficácia, tanto quanto à estrutura quanto aos processos.

As diversas metodologias de desenvolvimento de SI existentes enfocam especialmente a identificação de meios para aumentar a eficiência e eficácia de áreas funcionais específicas. Uma parcela se dedica à modelagem formal das áreas operacionais na identificação de possíveis ganhos. Abordagens como a Análise Estruturada (DeMARCO, 1989), o BSP - Business Systems Planning (IBM, 1981) e OAM - Office Analysis Methodology (SIRBU, 1981) são exemplos representativos aplicáveis a funções fortemente estruturadas. Para as funções menos estruturadas, a técnica FCS - Fatores Críticos de Sucesso (ROCKART, 1979) tem obtido bons resultados.

Entretanto, apesar desses bons resultados, essas metodologias não estão direcionadas para a

identificação de oportunidades do uso de recursos de informação em um enfoque estratégico. Seu conteúdo está restrito a uma visão organizacional de acordo com técnicas tradicionais de SI. A Engenharia da Informação, de Martin e Finkelstein (MARTIN, 1985), propõe o planejamento estratégico de necessidades de informação, mas não é clara quanto aos métodos de avaliação do valor dos possíveis benefícios. Essas técnicas são consideradas como instrumentos de domínio da organização, independentes da metodologia em si.

2.2. Recursos de Informação e Estratégia Competitiva

A estratégia competitiva se insere no ambiente em que a organização atua. O papel da empresa, nesse contexto, é gerar oportunidades que determinem ganhos para adquirir vantagem competitiva. Os RI têm função importante nesse processo, pois deles depende toda a comunicação da empresa com o seu meio.

Dois modelos de geração de oportunidades de estratégia competitiva são apresentados a seguir como exemplos de como as empresas podem utilizar os RI no seu negócio.

2.2.1. Modelo das quatro áreas de oportunidade

Bakopoulos e Treacy (BAKOPOULOS, 1985a) sintetizam o trabalho de diversos autores identificando quatro áreas de oportunidade para que os recursos de informação apóiem a estratégia competitiva: (a) aumento da eficiência operacional e da eficácia funcional; (b) inovação de produtos através de recursos de informação; (c) aumento do poder de barganha em relação a clientes e fornecedores; e (d) exploração de sinergias inter-organizacionais.

O aumento da eficiência operacional e da eficácia funcional é o foco tradicional de aplicação dos recursos de informação no suporte da estratégia interna da empresa. Esses sistemas podem conduzir a vantagens competitivas, desde que representem inovações significativas no processo

produtivo, diferenciando a organização em relação às outras.

A inovação de produtos através de recursos de informação podem ser identificadas de duas formas. Primeiro, pela incorporação de RI a produtos para aumentar o seu valor. Segundo, pela utilização de RI como um veículo de novos produtos baseados em serviços.

O aumento do poder de barganha em relação a clientes é obtido através do fornecimento de informações e serviços distintos que impliquem custos ao comprador numa eventual troca por um concorrente. Essa situação também deve ser percebida quando a empresa está na posição de cliente em relação a um fornecedor. Dessa forma, ela deve evitar a dependência de serviços e informações controladas por um fornecedor e criar um eficiente *mercado eletrônico* entre a empresa e seus fornecedores.

Uma aplicação recente desse mercado eletrônico é o EDI (Eletronic Data Interchange), uma rede de comunicação que permite que negócios sejam realizados exclusivamente através da troca de informações por computador. É possível envolver diversas organizações num intercâmbio de informações, de maneira a otimizar o tempo das transações, programar melhor os estoques para a produção e negociar com maior precisão as questões financeiras. Desse modo, o EDI pode envolver a empresa compradora de uma mercadoria, o fornecedor, a transportadora e a seguradora do produto a ser distribuído, e o banco cobrador da operação.

O EDI pode ser um sistema proprietário (padrão adotado por uma empresa e utilizado pelos seus fornecedores), o que o torna, de certa maneira, restrito às organizações que possuem sistemas tecnicamente compatíveis. Para ampliar as possibilidades de comunicação, existem as *clearing houses* ou *redes de valor agregado*, que intermediam a troca de informações entre empresas, inclusive aquelas que possuem sistemas incompatíveis. Assim, mesmo já existindo uma convenção internacional que padroniza o EDI, ele pode ser considerado um sistema aberto, possibilitando um número muito maior de intercâmbio de informações entre empresas, gerando importantes benefícios para ambas as partes.

Pode-se considerar como vantagens proporcionadas pelo EDI, a redução de custos, a possibilidade de acessar novos mercados, a inovação de produtos e serviços, a diminuição dos riscos e da incerteza nos negócios e a otimização da comunicação no ambiente empresarial.

A exploração de sinergias inter-organizacionais, com clientes e fornecedores, geralmente se concentram em oportunidades de melhor coordenação. A coordenação pode ser obtida através de SI que acoplem áreas funcionais de duas empresas distintas. Esses sistemas inter-organizacionais permitem que as organizações se integrem verticalmente. Metodologias para identificar oportunidades para sistemas cooperativos podem ser bastante similares às utilizadas no aumento da eficiência operacional e eficácia funcional. Existe, porém, uma diferença quanto à unidade de análise, que deixa de ser apenas uma única organização para serem abordadas várias empresas.

É prática usual em países como Japão e Estados Unidos, hoje presente também no Brasil, que grandes empresas participem do controle de qualidade da produção de seus fornecedores. Essa participação é concretizada principalmente através do uso de RI, com transmissão de tecnologia, treinamento de RH, técnicas de administração da produção e de controle de qualidade. Os SI cumprem, portanto, importante papel nesse processo, promovendo ganhos de competitividade em ambos os lados do negócio.

2.2.2. Gerador de opções estratégicas

Wiseman e MacMillan (WISEMAN, 1989) propõem o *gerador de opções* como um instrumento para identificar oportunidades para obter vantagem competitiva através da utilização de RI. Além disso, geralmente é adicionado valor aos produtos e serviços oferecidos aos clientes.

Uma unidade de negócio utiliza uma série de cinco questões evocadas para gerar alternativas de oportunidade. As opções disponíveis para resposta não são mutuamente exclusivas, pois o tomador de decisão pode

escolher uma ou mais com graus de prioridade diferentes. A primeira pergunta diz respeito à escolha de um alvo estratégico para direcionar a vantagem do sistema de informações. Os alvos são fornecedores, clientes e empresas concorrentes:

- Qual o alvo estratégico ?

  . Fornecedores de matérias-primas, capital, trabalho ou serviços;
  . Clientes que compram, distribuem ou utilizam os produtos vendidos;
  . Concorrentes que estão atualmente no ramo de atividade, possíveis iniciantes, empresas em outras áreas oferecendo produtos substitutos ou outras organizações lutando por recursos escassos.

Para cada alvo existe um tipo de aplicação diferente para tornar o sistema de informações uma arma competitiva. Para atingir os fornecedores, os sistemas mais visados são os de compra e estoque. Tendo o cliente como alvo, o investimento deve ser feito em sistemas que agilizem o serviço, melhorem o nível de atendimento e a qualidade do produto. Para enfrentar a concorrência, são usuais os sistemas que auxiliam no desenvolvimento de novos produtos e serviços. Há ainda os sistemas que possuem alvos múltiplos, utilizados para atividades de suporte num determinado local, mas que proporcionam vantagem competitiva em outra área.

A partir da escolha do alvo (fornecedor, cliente e/ou concorrente), o *gerador de opções* pergunta de que forma atingi-lo. As alternativas sugeridas são pela diferenciação, pelo custo e pela inovação:

- Que abordagem estratégica pode ser utilizada no alvo ?

  . Diferenciação: reduzir ou aumentar vantagens de diferenciação em relação a clientes, fornecedores e concorrentes;
  . Custo: reduzir ou evitar custos em relação aos fornecedores, clientes ou concorrentes, ou auxiliar seus fornecedores a reduzir custos para receber tratamento preferencial ou aumentar os custos de seus

concorrentes;

Inovação: encontrar novas formas de realizar negócios através da utilização de recursos de informação.

Como diferenciação, os sistemas que substituem a produção humana reduzem as vantagens de exclusividade das atividades dos trabalhadores. Por outro lado, mesmo que a empresa preste serviços ou venda produtos similares à concorrência, a existência de um sistema que forneça informações aos seus clientes gera diferenciação e aumenta sua vantagem competitiva. Outra alternativa é o uso de sistemas orçamentários e de planejamento que possibilita melhor previsão de gastos e, conseqüentemente, deve reduzir custos. Fornecer informações detalhadas de um bem ou serviço gera maior segurança para sua aquisição pelo cliente. Por fim, sistemas de controle auxiliam na decisão de reduzir ou aumentar preços, escolher a melhor forma de realizar um negócio ou mesmo criar novas atividades para atingir novos mercados.

A escolha do alvo e da abordagem estratégica para atingi-lo resulta em nove opções de adquirir vantagem competitiva. Dentro dessa série de combinações, é preciso definir de que modo será empregado o sistema de informações: ofensiva ou defensivamente.

- Que modo estratégico pode ser utilizado ?

  . Ofensivo: aumentar uma vantagem;
  . Defensivo: reduzir uma vantagem detida por um dos alvos.

Para selecionar a direção de abordagem a ser utilizada, executivos de linha e de sistemas devem estar envolvidos no desenvolvimento de recursos de informação estratégicos. A direção pode ser o uso ou a provisão dos SI:

- Que direção de abordagem pode ser utilizada ?

  . Uso: os recursos de informação serão utilizados pela organização;
  . Provisão: os recursos de informação serão

fornecidos pela organização.

O último conjunto de opções para gerar vantagem competitiva está na escolha de capacidades dos SI que, basicamente, são processamento, armazenamento e transmissão de informações:

- Que capacidades dos Sistemas de Informação podem ser utilizadas ?

. Processamento;
. Armazenamento;
. Transmissão.

Apesar da simplicidade no uso do *gerador de opções* e dos resultados obtidos por sua aplicação, é necessário salientar que seu uso em organizações exige uma postura gerencial de compreensão da necessidade do planejamento estratégico. Sua adoção e prática é que configuram a maior validade do uso do *gerador de opções*. Nessa situação, a organização ganharia no estabelecimento de planos táticos das áreas operacionais e no aprendizado obtido pelos participantes do processo quanto às necessidades e estratégias organizacionais.

2.3. Recursos de Informação e Estratégia de Portfólio de Negócio

Porter e Millar (PORTER, 1985) sugerem as seguintes questões para a identificação de oportunidades de novos negócios:

- Quais as informações geradas (ou potencialmente geradas) no negócio a empresa pode vender ?

- Qual a capacidade de processamento de informação existe internamente para começar um novo negócio ?

- Os recursos de informação permitem a produção de novos itens relacionados aos produtos da empresa ?

Os RI pensados como produto ou parte deste, no mínimo reposiciona a visão organizacional da área de SI. A área de processamento de dados é parte integrante e vital para o futuro das organizações.

Segundo Bakopoulos e Treacy (BAKOPOULOS, 1985a), os recursos de informação têm alterado as fronteiras de eficiência dos processos produtivos das organizações, a tal ponto que novos mercados têm sido criados e outros têm desaparecido.

Esses impactos na estrutura industrial criam oportunidades das empresas aumentarem seu portfólio de negócios. A exploração dessas oportunidades requer um processo suficientemente sofisticado de formulação de estratégias que possibilitem identificá-las e, ao mesmo tempo, a capacidade de tornar disponível a tecnologia apropriada. Ou seja, é necessário ligar a formulação de estratégias com o desenvolvimento de tecnologia. O aspecto mais importante dessa ligação é que a estratégia da empresa forneça uma direção para a construção de uma base tecnológica. Outra alternativa seria a identificação do aproveitamento da superioridade tecnológica existente em novos negócios.

A implementação dessas estratégias requer o uso eficiente dos RI, o que se constitui numa tarefa complexa Portanto, a habilidade de explorar esses recursos para adquirir vantagem competitiva está sujeita a fatores que inibem e fatores que facilitam a sua utilização.

3. FATORES INIBIDORES E FACILITADORES NO USO DOS RECURSOS DE INFORMAÇÃO PARA ADQUIRIR VANTAGEM COMPETITIVA

Bakopoulos e Treacy (BAKOPOULOS, 1985a) relacionam como causas da subutilização dos recursos de informação para vantagem competitiva a:

- ignorância da alta administração sobre os recursos de informação e seus usos potenciais;
- comunicação fraca entre grupos de SI e outras áreas da empresa;
- resistência a mudancas, tanto na área de sistemas de informação como em outras áreas da empresa,
- falta de um objetivo dirigido a oportunidades para vantagem competitiva;

- falta de medidas eficientes de impacto, que acabam inibindo os investimentos.

King, Grover e Hufnagel (KING, 1989), a partir de pesquisa realizada junto a membros da Society of Information Management - SIM, constataram os seguintes fatores inibidores como os mais importantes no uso de RI:

- outras prioridades como sendo mais importantes;

- dificuldade de mensurar benefícios tangíveis;

- falta de planejamento apropriado;

- falta de suporte organizacional e da alta administração;

- baixa percepção da importância do conceito;

- restrições orçamentárias.

Por outro lado, os principais fatores facilitadores também foram identificados:

- forte suporte técnico e conhecimento dentro da empresa;

- disponibilidade de recursos computacionais;

- posição de liderança quanto à tecnologia disponível;

- pressão por competição;

- forte posição financeira da empresa;

- posição firme no mercado;

- estrutura organizacional e alta administração fortes;

- necessidade de ser único ou de inovação.

Analisando todos esses fatores, pode-se depreender que a utilização de RI para se obter vantagem competitiva depende fundamentalmente da visão de negócio da alta administração das empresas. Existem claros obstáculos quanto ao aproveitamento ideal do potencial existente desses recursos, causados principalmente pelo descompasso entre o grande avanço tecnológico na área de RI e a velocidade de absorção desses conhecimentos. Por outro lado, verifica-se que o domínio desses tecnologias não significa, por si só, a

existência de pré-requisitos necessários à obtenção de qualquer tipo de vantagem. Há, portanto, a necessidade de uma visão abrangente para se estabelecer políticas e práticas que possibilitem o aproveitamento estratégico dos RI.

CONCLUSÃO

A exploração dos recursos de informação pode significar fortes ganhos estratégicos e, como arma dos negócios, representa um desafio de várias dimensões para as empresas.

A detecção de oportunidades depende basicamente da disseminação de um processo criativo por toda a estrutura organizacional, afetando a natureza do trabalho e a sua gestão.

Nesse contexto são constatados vários aspectos como a necessidade de um comprometimento global com a estratégia da organização, métodos deficientes de integração de áreas funcionais e uma quase inexistente cultura de planejamento estratégico em setores tradicionalmente voltados ao âmbito operacional. Além disso, verifica-se o agravante da cultura especializada das áreas chave envolvidas. Mais do que nunca há a necessidade de profissionais hábeis no tratamento dos negócios da empresa atuarem com objetivos dirigidos à competitividade.

Mesmo que o ambiente econômico onde a empresa se insere não pressione à competição, é necessário vislumbrar que o preparo para o oposto é tarefa de lenta execução, pois implica significativas e radicais mudanças. Uma das principais, no que tange a SI, é de possuir internamente o domínio das tecnologias de informação mais atualizadas, para amenizar e tirar o melhor proveito dos efeitos de substituição tecnológica. O aprendizado contínuo é, sem dúvida, procedimento estratégico fundamental.

Um dos grandes desafios é o de adequar a formação dos recursos humanos à tecnologia existente nas áreas envolvidas ou no trabalho em grupo. De qualquer

maneira, faz-se necessária uma revisão na política de recursos humanos, visando criar uma cultura tecnológica básica e um ambiente que possibilite a inovação permanente. Uma das formas de promover um entendimento generalizado na utilização dos recursos de informação seria o incentivo aos processos ligados à *computação pelo usuário final*.

Outra importante conseqüência seria a revisão do papel da área de processamento de dados na organização. Num ambiente em que os RI devem ser considerados como uma arma, é fundamental descentralizar a organização da área de SI, colocando-a sob responsabilidade da alta administração e integrada ao planejamento estratégico da empresa (influenciando e sendo influenciada por ele).

A cultura gerencial das organizações tradicionais está em cheque. Há a necessidade de uma administração que valorize cada vez mais o indivíduo, sua formação profissional e um ambiente cooperativo. Técnicas gerenciais apropriadas precisam ser introduzidas para gerar o ambiente desejado.

No âmbito da pesquisa, é importante a necessidade de integrar as diversas bases teóricas, tanto as organizacionais, que compreendem as áreas envolvidas, quanto as técnicas, que criam os modelos de planejamento estratégico dos recursos de informação num contexto de negócio.

Uma das principais conseqüências nessa área parece ser a necessidade de rever as metodologias de desenvolvimento de sistemas e adequá-las às necessidades da competição. Além de introduzir métodos que permitam a detecção de oportunidades e de mensuração mais apurada de benefícios, é importante refletir sobre a pertinência dos métodos baseados nos analistas *especialistas*. É possível ser especialista em tudo o que está envolvido? Parece que há a indicação por metodologias que, além de possibilitarem o desenvolvimento de sistemas, facilitem o aprendizado, a participação e a integração de todas as áreas relacionadas.

Assim, o aproveitamento competitivo dos RI parece ser uma questão que não pode ser desprezada

pelas organizaçoes. Cabe aos seus administradores a percepção da necessidade de adquirir cultura sobre a área de informática. Cabe também, por outro lado, que os executivos de sistemas aprendam sobre a cultura e os valores organizacionais. Ambos têm o desafio de viabilizar a adoção de medidas para a utilização dos recursos de informação para adquirir vantagem competitiva.

REFERÊNCIAS BIBLIOGRÁFICAS:

BAKOPOULOS, J.A.Y. e TREACY, M.E. Information technology and corporate strategy: a research perspective. Sloan School of Management, working paper 1639-85, March, 85.

BAKOPOULOS, J.A.Y. Toward a more precise concept of information technology. In: 6th International Conference on Information Systems, 1985. Aeud: KING, 1989.

BENJAMIN, R.I., ROCKART, J.F., MORTON, M S.S. e WYMAN, J. Information tecnology: a strategic opportunity. Sloan Management Review, Spring, 1984.

BRANCHEAU, J.C. e WETHERBE, J.C. Key issues in information systems management. MIS Quarterly, (11:1), March, 1987.

CLEMONS, E.K. Information systems for sustainable competitive advantage. Information & Management, 11, 1986.

DeMARCO, T. Análise estruturada e especificação de sistema. Campus, Rio de Janeiro, 1989.

FENSTERSEIFER, J.E. Um modelo conceitual para avaliação de investimentos em tecnologias flexíveis de produção. Documento para estudo, PPGA/UFRGS, Porto Alegre, Novembro, 1990.

IBM Corporation. Business systems planning - information systems planning guide. Application manual GE20-0527-3. IBM Corporation, Third edition, July, 1981.

KING, W.R., GROVER, V. e HUFNAGEL, E.H. Using information and information technology for sustainable competitive advantage: some empirical evidence. Information & Management. 17 (1989).

MARTIN, J. e FINKESTEIN, C. Engenharia da informação: técnicas e abordagens de implantação. Compucenter, Rio de Janeiro, 1985.

PORTER, M.E. e MILLAR, V.E. How information gives you competitive advantage. Harvard Business Review, July-August, 1985.

MENDES, C.D. Informática e competitividade da empresa. In: XX Congresso Nacional de Informática, 1987.

OLIVEIRA, A.C.M.C. e GRAJEW, J. O enfoque do valor adicionado:informática e aumento de competitividade. In: XX Congresso Nacional de Informática, 1987.

ROCKART, J. F. Critical success factors. Harvard Business Review, 57(2), March-April, 1979.

SIRBU, M., SCHOICHET, S., KUNIN, J. e HAMMER, M. OAM: an office analysis methodology. Office Automation Group Memo OAM 016, Massachussets Institute Technology, 1981.

WISEMAN, C. e MacMILLAN, I.C. Criando armas competitivas através dos sistemas de informação. MIS - Relatório de Gerenciamento da Informação, (ii), Fevereiro, 1989.

ANÁLISE DA METODOLOGIA JAD - JOINT APPLICATION DESIGN - DE MODELAGEM DE SISTEMAS: DO PARADIGMA À APLICAÇÃO

Ricardo Teitelroit e Luiz Henrique Boff
Mestrandos do Programa de Pós-graduação em Administração
Área de Produção e Sistemas
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Av. João Pessoa, 52 - 1º. andar - Sala 11
90040 Porto Alegre(RS) - BRASIL
Telefone: (0512)25-6533 - Fax: (0512)25-5253

RESUMO

As transformações nas abordagens de gestão empresarial têm evoluído muito rapidamente nos últimos anos. Novos enfoques são adotados nos processos envolvidos, alterando sobremodo a utilização da informação. Os sistemas de informação estão cada vez mais mostrando seu caráter interdisciplinar e interfuncional, exigindo dos profissionais de processamento de dados mudanças em sua forma de atuar e de se integrar às novas necessidades.

Este trabalho visa analisar a metodologia de modelagem de sistemas JAD - Joint Application Design - desde seus fundamentos teóricos a seus requisitos de implantação. São apontadas questões paradigmáticas, problemas do processo de desenvolvimento de sistemas, características da metodologia e considerações a partir da observação da aplicação de dois casos reais.

ABSTRACT

Managerial approaches has evolved rapidly in the last years. New processes are altering the utilization of information. Information Systems have interdisciplinary and interfunctional characteristics. That demands changes in the way Information Systems professionals work.

This paper aims analysis of the methodology JAD - Joint Application Design - both from a theoretical point of view and its implementation. Questions, process problems, systems development, characteristics of methodology and considerations are all made from observations of two cases.

INTRODUÇÃO

O desenvolvimento de sistemas de informação (SI) e suas metodologias tem sido alvo de diversos trabalhos acadêmicos e experiências de implementação nas organizações. Na segunda metade da década de 70, observou-se o surgimento de diversas abordagens metodológicas. A década de 80 foi marcada especialmente pela adoção de algumas metodologias mais voltadas a atender as necessidades organizacionais.

Trabalhar com informática passou a ser uma preocupação generalizada a todos os tipos de organização. A área de SI, sobretudo nas grandes empresas, destaca-se principalmente pela de concentração de investimentos em recursos humanos e em inovação tecnológica.

Simultaneamente à evolução das técnicas de engenharia de sistemas e sua implantação, o ambiente econômico mundial tem sofrido grandes transformações. Enfatizam-se os conceitos de competitividade, qualidade e visão estratégica de negócio. Surgem cada vez mais aplicações da área de informática na estratégia competitiva das organizações.

Dentre os desafios surgidos, destaca-se a necessidade de reavaliar a pertinência do emprego das metodologias de desenvolvimento existentes e sua adequação às novas formas de gestão emergentes.

O presente trabalho compreende a análise da aplicação da metodologia Joint Application Design (JAD) de modelagem de sistemas, a partir de alguns pressupostos teóricos que serão expostos a seguir, buscando evidenciar as particularidades quanto ao processo que a abordagem propõe.

A JAD, desenvolvida inicialmente por um grupo liderado por Chuck Morris, da IBM do Canadá, foi adaptada por Hugo Fenido Gattoni, da IBM, filial Belo Horizonte, para utilização no Brasil (IBM, 1988). As considerações quanto à sua aplicação se baseiam em dois casos práticos observados.

A primeira parte do trabalho é uma contextualização do desenvolvimento de SI a partir de dois pressupostos: um trata dos princípios implícitos gerados durante o processo de desenvolvimento de sistemas; o outro, sobre a problemática e os conceitos metodológicos

relacionados com a engenharia de SI. A segunda parte do artigo descreve sucintamente a metodologia em questão, seus objetivos, etapas e requisitos envolvidos. A terceira parte aborda a análise da aplicação da metodologia e dos resultados obtidos. Finalmente, conclui-se com críticas e sugestões quanto à estrutura e aplicabilidade das ferramentas.

1. CONTEXTO DO DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO

Torna-se cada vez mais importante refletir sobre as diferentes abordagens existentes de desenvolvimento de SI. Todas elas, sem exceção, são constituídas de uma visão de mundo. é um conjunto de idéias básicas que definem e orientam a tomada de posição, sem neutralidade, para as atividades decorrentes do processo.

São particularmente instigantes algumas visões, mais contemporâneas, que têm evidenciado a fundamental importância dos aspectos sociais vinculados ao desenvolvimento de SI. Duas dessas abordagens são descritas como pressupostos teóricos de base para a análise da metodologia JAD.

1.1. Paradigmas de desenvolvimento de sistemas de informação

O conjunto de princípios adotado por uma comunidade profissional, que permite aos seus membros partilharem percepções similares e se engajarem em práticas comuns, segundo Hirschheim e Klein (HIRSCHHEIM, 1989), é chamado de *paradigma*. Esses princípios são relativos ao conhecimento e o modo como ele é adquirido (princípios epistemológicos) e sobre o mundo físico e social (princípios ontológicos).

Burrel e Morgan (BURREL, 1979) partem desses princípios para definirem duas dimensões paradigmáticas: uma *subjetivista-objetivista* e outra de *ordem-conflito*.

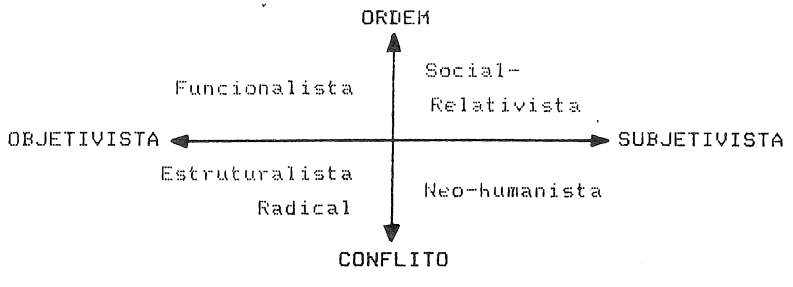
A posição *subjetivista* nega a propriedade dos métodos das ciências naturais para estudar o mundo social. Seu objetivo é compreender a vida humana pela experiência subjetiva dos indivíduos. A preocupação principal é com a

compreensão da forma pela qual os indivíduos criam, modificam e interpretam o mundo em que se encontram. A essência da posição *objetivista* é aplicar modelos e métodos derivados das ciências naturais para estudar questões humanas. Trata o mundo social como um mundo natural.

A posição da *ordem*, ou integracionista, enfatiza o mundo social como caracterizado pela estabilidade, integração, consenso e coordenação funcional. Já a posição de *conflito*, ou coerção, enfoca a mudança e a desintegração ambiental.

Essas duas dimensões, colocadas ortogonalmente, geram quadrantes que representam quatro paradigmas no desenvolvimento de sistemas (figura 1.1):

Figura 1.1 - Paradigmas de desenvolvimento de SI



Fonte: HIRSCHHEIM, 1989 (adaptado de BURREL, 1979).

A visão *funcionalista* busca explicar o status quo, a ordem e a integração social, o consenso, a satisfação de necessidades e a escolha racional. Tenta definir como os elementos individuais interagem para formar um todo integrado. A visão *social-relativista* procura explicações no terreno da subjetividade individual, e dentro de um quadro de refefencia do ator social como oposto ao observador da ação. Os "papéis sociais e instituicionais existem como uma expressão de significados que os homens atribuem a seu mundo". A visão *estruturalista radical* enfatiza a necessidade de transcender as limitações colocadas pelos arranjos sociais e organizacionais existentes, com foco primário na estrutura e análise dos relacionamentos de poder

econômico. Por fim, a visão *neo-humanista* busca a mudança radical e a emancipação, afirmando os diferentes papéis que as forças sociais e as organizações executam na compreensão das mudanças. Enfoca barreiras como ideologia, poder, compulsões psicológicas e restrições sociais, buscando caminhos para superá-las.

Para o escopo deste trabalho, somente serão caracterizados com maior profundidade os paradigmas funcionalista e social-relativista.

A abordagem mais tradicional no desenvolvimento de SI parece ser a funcionalista, onde o analista atua como um "especialista". Nessa visão, todo sistema é desenvolvido para contribuir com determinados fins específicos, geralmente definidos pela alta administração das organizações. A solução de polêmicas e eventuais conflitos é atribuição exclusiva da direção da empresa. Os métodos e os meios estão preocupados em escolher as melhores ferramentas para atingir os fins. O papel do analista é o de um especialista em tecnologia, ferramentas e métodos de desenvolvimento de SI e administração de projetos. Todo o processo de desenvolvimento de sistemas acontece de maneira formal e racional, com pouca relevância à intuição humana, aos julgamentos e aspectos políticos (considerados irracionais e fatores de interferência na eficiência e eficácia do processo).

Segundo Hirschheim e Klein (HIRSCHHEIM, 1989), a alta administração define os objetivos, cabendo ao analista transformá-los em produtos (o sistema), e aos usuários operarem ou interagirem com o sistema para serem concretizados os planos da organização. Nesse caso, a epistemologia é a aquisição do conhecimento e se dá pelos relacionamentos mensuráveis entre causa e efeito (abordagem positivista). A ontologia baseia-se no realismo, a partir da visão de que a realidade empírica da organização existe independentemente do observador.

Conforme Burrel e Morgan (BURREL, 1979), o funcionalismo busca fornecer, essencialmente, explicações racionais às questões sociais. Aspectos como poder, conflito de interesses e objetivos do sistema fogem do domínio do analista.

Os problemas atribuídos a essa abordagem são apontados como:

- a ênfase aos meios sem discutir os fins;

- a legitimidade dos fins pode estar baseada no uso do poder;

- as questões econômicas são governadas por convenções sociais e por decisões de elites sócio-políticas;

- as pessoas têm escolha livre e a observação não é neutra, influindo no processo de desenvolvimento.

A abordagem social-relativista encara o desenvolvimento de SI como "produção de significado". Partindo da concepção de que o negócio não se restringe a uma realidade econômica objetiva, e que há diferentes percepções sobre a realidade, existe uma busca para atingir o consenso e a legitimação dos objetivos através do envolvimento de todas as partes envolvidas. O projeto do sistema resulta de uma interação social, reduzindo as dificuldades de mudança cujos propósitos são desconhecidos por todos.

Os usuários são agentes organizacionais geradores de significado. O analista é uma agente de mudanças, utilizando sua experiência para facilitar o processo sem introduzir questões substantivas. Assim, o desenvolvimento de SI cria novo significado através do interacionismo simbólico.

O conhecimento (epistemologia) é obtido pela produção de significado. A nível ontológico, concebe-se a realidade como algo mutável socialmente construído, como produto da mente humana. Os significados emergentes são produtos da experiência que está sempre mudando e nunca é igual para duas pessoas.

O objetivo do analista é o de facilitar o aprendizado do todo que está envolvido ou sendo afetado. Ele ajuda a estimular a reflexão, a cooperação e a aquisição de conhecimento experimental. Geralmente são utilizadas ferramentas para auxiliar a interação do grupo do projeto.

Há a crença de que o resultado do desenvolvimento do SI não é tão importante quanto o processo para atingi-lo. Existe uma forte participação, um estímulo à criatividade e produção de significado. A aceitação durante o processo de

desenvolvimento é a única premissa que realmente interessa. Portanto, a interação social é crucial para a aceitação. Não há, contudo, espaço para a distinção entre consenso válido ou falacioso.

A metodologia não propõe nenhuma abordagem crítica, em relação aos efeitos colaterais disfuncionais conseqüentes de sua aplicação, em relação ao uso de ferramentas e técnicas para o desenvolvimento de SI. Diferentes produtos do processo são vistos como resultantes de diferentes realidades socialmente construídas.

1.2. A engenharia de sistemas de informação

Segundo Bracchi e Pernici (BRACCHI, 1984), um elemento fundamental de uma metodologia de desenvolvimento de sistemas de informação de escritórios é o tipo de visão conceitual adotada na fase de análise. Essas visões podem ser:

- *técnica*: analisa detalhadamente o trabalho de escritório, considerando as operações que são realizadas, medindo-as principalmente em termos de tempo de execução. Objetiva identificar métodos ótimos de realização das tarefas, evidenciando os aspectos físicos envolvidos em detrimento da natureza intelectual. É restrita quanto a uma visão de performance global da organização, por deter-se na mensuração de produtividade de tarefas isoladas e nos dados manipulados pelas mesmas.

- *organizacional*: é considerada a estrutura organizacional global e os objetivos de negócio. A estrutura hierárquica da organização é refletida no modelo de escritório.

- *sócio-técnica*: considera as tarefas desempenhadas por cada unidade em que a empresa é subdividida. Cada uma exerce algum controle pré-estabelecido sobre o trabalho. As ações tomadas se apóiam nos recursos e memória de cada unidade. A análise do trabalho do escritório é vista como um conjunto de atividades executadas por diversos grupos, onde as pessoas envolvidas são consideradas em conjunto com os recursos disponíveis. Portanto, há uma preocupação de otimização integrada dos aspectos técnicos e sociais.

Assim como as organizações e seus processos se modificam ao longo do tempo, os SI evoluem no sentido de atender a novas necessidades que surgem. Esse processo de mudança caracteriza o início do que é denominado como ciclo de desenvolvimento de sistemas.

Wetherbe (WETHERBE, 1987) visualiza e conceitua os componentes básicos do ciclo de desenvolvimento de sistemas como sendo:

- *Surgem Novos Problemas e/ou Oportunidades*: mudanças no contexto organizacional implicam novas necessidades ou melhorias em relação ao sistema de informações existente;

- *Análise do Sistema Existente*: essa fase consiste das seguintes atividades:

(a) rever o fluxo de trabalho;

(b) definir a tomada de decisão associada com o fluxo de trabalho;

(c) rever o fluxo de informação que apoia a tomada de decisão;

(d) isolar deficiências do sistema de informação. Essas deficiências podem ser de inclusão (ausência ou presença, no sistema, de informação, tecnologia ou pessoal) e de estrutura (organização e inter-relação, no sistema, de informação, tecnologia ou pessoal).

- *Soluções de Projeto*: são projetadas soluções para as deficiências detectadas, buscando aperfeiçoamento do fluxo de trabalho e/ou das decisões que estão sendo tomadas.

- *Identificação de Meios para as Soluções*: são identificadas as tecnologias viáveis e o pessoal necessário para a obtenção das soluções definidas na etapa anterior.

- *Avaliação e Seleção dos Melhores Meios para as Soluções*: são analisadas as soluções possíveis e escolhida a mais adequada.

- *Desenvolvimento e Teste do Sistema*: estruturação real do sistema, tendo como produto uma aplicação operável e viável que atinja os objetivos do projeto original.

- *Implementação do Sistema*: processo de substituição do sistema antigo pelo novo.

- *Avaliação e Manutenção do Sistema*: verificação

da eficiência e eficácia do sistema em operação, em relação às soluções dos problemas e oportunidades identificados.

Durante o ciclo de desenvolvimento, segundo DeMarco (DeMARCO, 1989), o produto mais importante da análise de sistemas é o documento de especificação, que estabelece os objetivos para o restante do projeto. Esse documento, abrangeria os resultados obtidos nas etapas anteriores ao Desenvolvimento e Teste do Sistema do modelo definido por Wetherbe.

DeMarco (DeMARCO, 1989), ao avaliar as características da análise de sistemas, argumenta que a formalização desse documento de especificação é uma tarefa complexa, dada a heterogeneidade dos usuários e o conflito de interesses, exigindo do analista habilidades no tratamento de processos de negociação. Outros fatores que afetam esta etapa são as relações interpessoais e a falta de precisão da análise em si (inclusive para determinar o término dessa fase).

Os principais problemas que ocorrem na fase de análise, são apontados por DeMarco (DeMARCO, 1989) como sendo:

- *problemas de comunicação*: dificuldade natural de descrever procedimentos, inadequação dos métodos utilizados para essa descrição, falta de uma linguagem comum entre analista e usuário e a falta de um modelo prévio que possa ser utilizado para o sistema;

- *natureza variável dos requisitos do sistema*: a partir da especificação até a efetiva implementação, as mudanças continuam ocorrendo, havendo a necessidade de que o documento de especificação suporte alterações para se manter adequado;

- *falta de ferramentas adequadas*: a natureza subjetiva do processo de análise leva a um apelo à sabedoria e julgamentos do analista como sendo suas únicas ferramentas de avaliação;

- *problemas do documento de especificação*: normalmente são excessivamente redundantes, prolixos, físicos, proporcionam uma leitura tediosa e são insuportáveis para serem escritos;

- *distribuição de trabalho*: dimensionamento

inadequado de recursos e tempo da fase análise;

- *política*: a situação política do analista é complicada pelas falhas de comunicação e/ou pela inadequação de seus métodos. A causa mais evidente de dificuldades políticas está na distribuição mutável de força e autonomia que acompanham a introdução de um novo sistema.

2. A METODOLOGIA JAD - JOINT APPLICATION DESIGN

A metodologia JAD foi criada para acelerar o desenvolvimento de sistemas, utilizando uma abordagem estruturada e técnicas de dinâmica de grupo (SCHEIN, 1965) aplicadas a reuniões de projeto, com a participação de pessoas de uma ou mais áreas usuárias e de processamento de dados.

As aplicações normalmente atendidas pela metodologia envolvem banco de dados, projetos interativos, de média complexidade ou aplicações que usarão um determinado software (programa produto). A JAD é bastante flexível, possibilitando ser utilizada também em aplicações complexas. Nesses casos, primeiramente trata-se a aplicação como um todo e, na seqüência, são realizadas novas sessões abordando diversas partes específicas.

Conforme Gane (GANE, 1988), existem três tipos básicos de reuniões de projeto: a) sessões estratégicas, onde são discutidos o âmbito, objetivos, recursos, políticas e mudança organizacional; b) sessões de dados e processos, onde se constrói ou aperfeiçoa os diagramas de fluxo de dados e modelos de dados e; c)sessões de telas e relatórios, onde se definem as entradas e saídas dos sistemas.

A metodologia deve ser entendida como uma proposta inicial, podendo ser adaptada ao caso e ao ambiente onde está sendo aplicada.

2.1. Objetivos da metodologia

Os principais objetivos da JAD são:

- redução de tempo no desenvolvimento de aplicações;

- aumento de produtividade através da interação proposta entre os participantes das reuniões de projeto;

- melhor qualidade dos projetos desenvolvidos.

2.2. Etapas

A metodologia está dividida basicamente em três partes:

- *reunião inicial*: são estabelecidos os principais objetivos do projeto e planejado o trabalho;

- *reunião de revisão*: é feita uma revisão relativa à reunião inicial, avaliada a situação das tarefas previstas, identificados problemas e suas possíveis correções, buscando atingir a terceira etapa dentro dos prazos estabelecidos;

- *sessão de design*: são realizadas reuniões para desenvolver o projeto da aplicação enfocada.

2.2.1 Reunião inicial

Devem participar dessa reunião os responsáveis pelo estabelecimento dos objetivos e pelo planejamento dos trabalhos. Nela são tratados, entre outros, os seguintes assuntos:

- escolha da aplicação objeto;
- definição dos objetivos;
- definição dos participantes da sessão de design;
- definição dos convidados (pessoas que estão relacionadas com a aplicação objeto, atuando geralmente como observadores, mas que podem opinar e dar sugestões durante a sessão);
- data da reunião de revisão;
- data da sessão de design;
- distribuição de tarefas aos participantes da sessão de design (realizar um levantamento do organograma da empresa, documentos utilizados, volumes e fluxos do sistema atual, preparar o material de apoio para a sessão, entre outras);
- definição da necessidade de apresentação à área responsável pela aplicação sobre noções básicas de computação;
- definição do condutor da sessão de design;
- escolha do local para a sessão de design;

- visita ao departamento responsável pela aplicação;

- comunicar ao departamento responsável qual a aplicação que vai ser objeto do projeto;

- aviso da data da sessão de design ao pessoal do departamento responsável pela aplicação;

- início da criação do documento do projeto (ata da reunião inicial).

2.2.2. Reunião de Revisão

O objetivo da reunião de revisão é analisar os assuntos tratados na reunião inicial, para corrigir possíveis problemas e assegurar o bom andamento dos trabalhos. Além dos aspectos globais, devem ser verificados os seguintes pontos específicos:

- realização das tarefas designadas a cada um dos envolvidos no projeto;

- problemas de ordem burocrática que afetam a realização da sessão de design;

- problemas relacionados ao levantamento do fluxo do sistema atual;

- formalização de uma *agenda tentativa*, composta pelas atividades e tempos previstos da sessão de design;

- continuação dos trabalhos de documentação do projeto (ata da reunião de revisão).

2.2.3. Sessão de Design

Dessa etapa do trabalho, sem dúvida a mais importante do projeto, devem participar todas as pessoas escolhidas na reunião inicial e confirmadas na reunião de revisão. Normalmente, a cada atividade da sessão, são atribuídos tempos previstos, procurando determinar a quantidade necessária de reuniões.

A agenda tentativa, sugerida pela metodologia, prevê uma sessão de design com duração de dois e meio dias, composta pelas seguintes atividades:

- *abertura dos trabalhos:*

  a) apresentação dos participantes e convidados;

b) estabelecimento das datas e horários para a realização da sessão;

c) estabelecimento dos objetivos e escopo do trabalho;

d) apresentação do roteiro pretendido para o trabalho;

- *introdução à computação:*

a) características do equipamento;

b) formas de utilização e ferramentas disponíveis;

- *apresentação do organograma da empresa:*

a) apresentação;

b) distribuição do organograma aos participantes e convidados;

c) comentários, correções e confirmação do organograma;

- *recordação do fluxo atual das atividades:*

a) apresentação do fluxo atual;

b) distribuição do fluxo aos participantes e convidados;

c) comentários, correções e confirmação do fluxo atual;

- *problemas, defeitos e empecilhos do fluxo atual:*

a) apresentação dos problemas pelos participantes;

b) resumo dos problemas;

c) priorização dos problemas;

d) identificação dos problemas que podem ser resolvidos com o auxílio de processamento de dados;

e) consenso do grupo sobre os problemas, prioridades e soluções possíveis com PD;

- *definição dos objetivos:*

a) objetivos do trabalho (o que, como e para que fazer);

b) consenso do grupo;

- *racionalização do fluxo atual:*

a) modificações no fluxo atual, sem a ajuda de um sistema automatizado;

b) aprovação das racionalizações;

c) desenho do novo fluxo;

- *proposição do novo fluxo:*

a) adaptação do novo fluxo racionalizado à aplicações automatizadas;

b) aprovação do novo fluxo;

c) análise do novo fluxo (segurança);

d) modificações administrativas implicadas;

e) necessidades de hardware e software;

- *aprovação do novo fluxo:*

a) desenho do novo fluxo;

b) aprovação final do novo fluxo;

c) compromisso com o novo fluxo;

- *análise dos dados de entrada:*

a) verificação e validação dos volumes levantados;

b) dimensionamento das mensagens - transações;

c) acertos finais;

- *relatórios e telas:*

a) proposição de telas e relatórios desejados;

b) fac-símile de relatórios e telas;

c) validação das telas e relatórios;

- *definição do conteúdo dos arquivos:*

a) definição das necessidades de cada arquivo;

b) validação da definição dos arquivos;

- *cronograma - prazos e responsáveis:*

a) elaboração do cronograma de tarefas;

b) indicação dos responsáveis por tarefa;

c) validação dos responsáveis e prazos;

- *encerramento:*

a) distribuição do documento da sessão;

b) aprovação do documento;

c) assinatura pelos participantes:

d) avaliação da sessão de design;

e) encerramento dos trabalhos.

A metodologia sugere o uso de ferramentas de apoio durante a sessão de design, como recursos de comunicação visual - facilitando a compreensão dos participantes - e de edição de texto - para documentar e reproduzir o conteúdo detalhado dos assuntos que são discutidos.

3. CASOS E RESULTADOS

Como citado anteriormente, a metodologia foi aplicada em duas oportunidades em empresas do ramo da construção civil. O contexto dos sistemas analisados referia-se às áreas de planejamento, compras, execução de obras e apoio administrativo.

O nível de complexidade dos SI era o mesmo nas duas empresas. Porém, o grau de formalização das rotinas, o número de pessoas, níveis hierárquicos envolvidos e cultura de informática existentes foram bastante diferenciados.

Nas duas empresas, através de contatos com os responsáveis pelas áreas, obteve-se uma primeira visão do problema. Nas reuniões seguintes, foi proposta a aplicação da metodologia JAD, e então marcada a reunião inicial.

As datas das sessões de design foram submetidas à aprovação prévia de todos os participantes. Pela inviabilidade do uso do horário de expediente (apontada por todos os envolvidos), houve certa resistência e dificuldades de definição de datas, tendo sido adotado o máximo de um encontro por semana, fora da jornada de trabalho normal, nas dependências das empresas.

A partir daí todos os trabalhos observaram as propostas da metodologia, tendo transcorrido de forma fluente e logicamente bem encadeada. No decorrer dos encontros, os participantes de cada etapa demonstraram grande motivação e engajamento nas atividades desenvolvidas. Vencidas algumas inibições iniciais, o consenso foi obtido de forma objetiva através dos debates gerados.

O condutor da sessão - em ambos os casos, um analista de sistemas - atuou como um facilitador do grupo, buscando atingir os objetivos propostos. Com base nos levantamentos efetuados para a definição do fluxo atual, chegou-se a um primeiro modelo conceitual que serviu de referencial para a execução de cada etapa.

As ferramentas de apoio utilizadas na sessão de design foram:

- processador de textos: para a documentação dos assuntos tratados, possibilitando a distribuição de cópias aos participantes ao final de cada encontro;
- diagramas de decomposição funcional e diagramas

de fluxos de dados: para a representação dos fluxos de atividades;

- flip-chart e/ou quadro "branco": para a representação dos conteúdos abordados ao longo dos trabalhos.

Em ambos os casos, a metodologia foi aplicada de acordo com as etapas sugeridas, mas não foram abordados todos os passos constantes da agenda tentativa. No primeiro, a grande abrangência do sistema analisado indicou o desdobramento do problema em outras sessões de projeto, tendo sido suprimidas as etapas de análise dos dados de entrada, relatórios e telas e definição do conteúdo dos arquivos. Isso pode ser caracterizado como sendo do tipo estratégico, com estudo dos dados e processos. No segundo caso, atingiu-se a fase de definição dos objetivos.

Os principais resultados obtidos foram:

- formação de uma visão global da organização e de seus objetivos;

- reformulação e melhor entendimento da estrutura hierárquica e funcional;

- clareza quanto a necessidades e prioridades, no que diz respeito a SI e a outros recursos, em todas as áreas envolvidas;

- familiarização por parte dos participantes das técnicas de modelagem de sistemas utilizadas, facilitando a comunicação usuário/analista;

- otimização do processo de análise, no que se refere aos tempos e adequação da conceituação do problema e das soluções viáveis;

- no segundo caso, foram detectados sérios problemas em relação à estrutura organizacional e a questões políticas e administrativas, que apontavam ser inoportuna a implantação de sistemas, o que foi de alguma forma confirmado pela interrupção dos trabalhos;

- foi eliminada a necessidade de diversos encontros de validação de projeto com cada usuário;

- conhecimento e aprovação prévia da estrutura e objetivos do sistema proposto e comprometimento do grupo em relação ao produto da sessão;

- a representatividade do trabalho e do grupo que

gerou ' o projeto fortaleceu a obtenção dos recursos necessários junto a níveis hierárquicos superiores;

- os participantes manisfestaram grande motivação pelo tipo de trabalho realizado e pelos resultados obtidos, tendo se declarado gratificados pela oportunidade de atuar ativamente na determinação das mudanças propostas.

4. ANÁLISE CRÍTICA

A metodologia JAD caracteriza-se pela ênfase na dinâmica de grupo, utilizando-a como meio para abordar as questões sociais, organizacionais e técnicas envolvidas no desenvolvimento de aplicações. Pela abrangência dos problemas abordados, a participação dos usuários no processo, e pela emergência de questões relativas ao grupo de pessoas envolvidas, podemos classificá-la dentro do que Bracchi e Pernici (BRACCHI, 1988) conceituam como sendo uma abordagem sócio-técnica.

A JAD poderia ser enquadrada com base no paradigma social-relativista. Há, porém, que ser salientado que o arquétipo do "facilitador" proposto por Hirschheim e Klein (HIRSCHHEIM, 1989) pressupõe condições organizacionais de percepção da condição paradigmática. Para uma empresa cuja abordagem é predominantemente funcionalista, essa mudança não se efetivaria de forma muito natural e global, prejudicando os efeitos de iniciativas isoladas ou localizadas em determinados setores.

A aplicação da metodologia requer um entendimento e apoio da alta administração da organização e dos participantes do processo, sob pena de não gerar resultados adequados aos objetivos corporativos e ao grupo social envolvido. Há também o aspecto do poder atribuído à equipe no que diz respeito a modificações da estrutura organizacional e funcional. Mesmo não sendo objetivo dos trabalhos, o produto final representa as idéias e reinvidicações do grupo. Posteriores alterações ou contra-argumentos podem ser recebidos com muita resistência, e a não obtenção dos recursos para a continuidade do projeto é motivo de grande frustração.

Apesar de não ser associada pelos seus criadores

às fases do ciclo de vida do desenvolvimento de sistemas, a metodologia pretende cobrir as seguintes etapas conforme o modelo concebido por Wetherbe (WETHERBE, 1987): (a) surgimento de novos problemas e oportunidades; (b) análise do sistema existente; (c) soluções de projeto; (d) identificação de meios para soluções; (e) avaliação e seleção dos melhores meios para soluções. Seria preciso entender a metodologia como parte de um todo maior, o que não é referido em sua proposta. As ferramentas utilizadas poderiam ser adequadas à concatenação com uma metodologia que complementasse o ciclo, como, por exemplo, uma das propostas de a análise e projeto estruturado de sistemas.

Nota-se também que não há sugestões de ferramentas de apoio para os contatos iniciais do analista com o ambiente usuário. Nessa etapa, mostraram-se de bastante utilidade estudos de decomposição funcional e de tarefas e tempos envolvidos.

Encarando o produto resultante da sessão de design como o documento de especificação referido por DeMarco (DeMARCO, 1989), verifica-se que a JAD equaciona adequadamente os principais problemas apontados para a execução e formalização da fase de análise, ou seja:

- propõe um esquema estruturado de procedimentos e a utilização de ferramentas de comunicação, que reduzem as dificuldades de entendimento entre analista e usuários, e da documentação escrita dos trabalhos;

- o desenvolvimento conjunto dos processos de análise - mais do que uma ferramenta, uma filosofia de trabalho - reduz em muito a dependência quanto ao conhecimento e julgamentos da parte do analista, criando uma intersubjetividade que certamente resulta em melhor qualidade e promove a aprendizagem organizacional;

- o documento de especificação obtido não é redundante, prolixo ou físico, mas sim objetivo, estruturado, lógico, fácil de ser redigido e de ser entendido. Além disso, à medida em que se utilize sistematicamente a metodologia, esse documento tende a adquirir forma e estrutura padronizadas;

- o dimensionamento de tempos e recursos da fase de análise é extremamente facilitado, tendendo a diminuir os

erros de avaliação. Há a dificuldade de reunir os participantes quanto a datas e locais, podendo afetar os prazos de conclusão mas não o tempo dedicado ao trabalho. Nesse problema, é importante a atuação dos responsáveis pelas áreas envolvidas. Apesar de possibilitar uma aceleração no tempo de desenvolvimento de sistemas, os prazos envolvidos podem ser dilatados;

- à medida em que o papel do analista é o de um facilitador do processo, e que as soluções adotadas são debatidas e referendadas por todo o grupo, seus eventuais problemas políticos ficam bastante amenizados. Isso implica que o analista domine técnicas de dinâmica de grupo e possua boa capacidade de comunicação e liderança;

- a natureza variável dos requisitos do sistema não pode ser evitada, mas é de se esperar que pelo comprometimento obtido as mudanças necessárias sejam mais rapidamente percebidas e informadas.

Outro ponto importante a ser salientado é que ao longo do uso sistemático da metodologia, é possível que se diminua a necessidade de participação do analista de sistemas como facilitador, podendo este ser substituído pelos próprios usuários. Por outro lado, as organizações que incentivam a computação pelos usuários finais podem utilizar a JAD para a aprendizagem dos processos de análise e para a introdução de padrões.

CONCLUSÃO

Através deste trabalho, conclui-se que metodologia JAD vem preencher uma lacuna criada pela inexistência de técnicas adequadas para as fases de análise de sistemas. Como uma ferramenta bastante flexível, pode ser utilizada na abordagem de problemas com diversos graus de complexidade e na extensão desejada. Se adotada como rotina, constitui-se em importante instrumento para a implantação de tecnologia de informática nas organizações.

Talvez, o grande problema para sua aplicação esteja diretamente relacionado à formação de recursos humanos. O papel de facilitador proposto ao condutor da sessão de projeto, requer deste o domínio de técnicas que não são usuais aos analistas de sistemas. Além disso, requer

que a organização se conscientize da natureza do trabalho e que apoie sua execução. Empresas pequenas ou médias talvez encontrem dificuldades em manter em seus quadros profissionais com tais qualificações. Desta forma, estariam obrigadas a buscar ajuda externa, o que nem sempre é viável e desejável.

REFERÊNCIAS BIBLIOGRÁFICAS:

BRACCHI, G. e PERNICI, B. The design requirements of office systems. ACM Transations on Office Information Systems, 2(2), April, 1984.

DeMARCO, Tom. Análise estruturada e especificação de sistema. Campus, Rio de Janeiro, 1989.

HIRSCHHEIM, R. e KLEIN, H. K. Four paradigms of information systems development. Communications of the ACM, 32(10), October, 1989.

IBM do Brasil. JAD - Joint Application Design. Guia de implementação. IBM do Brasil, Belo Horizonte, 1988.

GANE, Chris. Desenvolvimento rápido de sistemas. Livros Técnicos e Científicos, Rio de Janeiro, 1988.

SCHEIN, E. e BENNIS, W. Personal and organizational change through group methods. Wiley, New York, 1965. Apud: GANE, 1988.

WETHERBE, James C. Análise de sistemas para sistemas de informação por computador. Campus, Rio de Janeiro, 1987.

Modelagem de Esquemas em um Ambiente de Banco de Dados Orientado a Objetos

Leonardo Chaib Rossetti

Ana Maria de C. Moura

Instituto Militar de Engenharia
Seção de Matemática e Engenharia de Sistemas
Pça General Tiburcio 80 - Praia Vermelha
CEP: 22290 - Rio de Janeiro - RJ
Brasil

RESUMO

O paradigma de Orientação a Objetos (O-O) veio de encontro às exigências dos atuais Bancos de Dados, pois permite a definição de entidades que incorporam às suas estruturas o respectivo comportamento, viabilizando a construção de sistemas adaptáveis às necessidades do mundo real, conforme já demonstram na prática alguns sistemas gerenciadores de Banco de Dados Orientados a Objetos (SGBDOOs). Este artigo apresenta o desenvolvimento de um ambiente de modelagem de um Banco de Dados Orientado a Objetos (BDOO), a partir de um modelo de dados baseado em rede semântica estendido a objetos.

I - INTRODUÇÃO

A evolução experimentada pela área de Banco de Dados (B.D.) nas duas últimas décadas foi fortemente influenciada pelo afluxo gradual dos inúmeros modelos desenvolvidos para projetos de B.D., refinados a cada nova metodologia proposta. Primeiramente vieram os modelos clássicos de dados : hierárquico, redes e relacional (CODD70). Apesar de terem atendido a grande parte das necessidades dos usuários, estes modelos procuravam representar basicamente como os dados eram fisicamente modelados, deixando escapar boa

parte da semântica da aplicação, quase sempre embutida nos programas dos usuários. Os modelos semânticos surgiram a seguir, com o objetivo de preencher esta considerável lacuna entre o nível externo, visão do usuário, e o nível interno. Entre os mais conhecidos destacam-se o ER (CHEN76), SAM*(SU83), TAXIS (MYLO82), MORSE (BOUZ86), GALILEO (ALBA85) e muitos outros. O principal avanço introduzido pelos modelos semânticos foi a abstração estrutural, a exemplo dos conceitos de generalização, classificação e agregação. Porém, carecendo do aspecto comportamental relativo às entidades modeladas, deixam grande lacuna no que diz respeito a manutenção da base de dados.

Apesar do avanço possibilitado por estes modelos, muitas limitações permaneceram, principalmente devido à estrutura física baseada em registros, que obrigava a representação de dados homogênea tanto horizontal como vertical, nem sempre desejada. Estas limitações agravaram-se ainda mais com o surgimento das aplicações ditas não convencionais, tais como : sistemas CAD/CAM, Automação de Escritórios, BD Textuais e Pictóricos, Sistemas Cartográficos, e outros que, por envolverem o processamento de dados complexos, embutidos muitas veses em um grande número de transações longas, podem ser enquadrados nesta categoria.

Segundo Dittrich (DITT86), uma base de dados pode ser considerada O.O se for capaz de representar qualquer entidade do mundo real, independentemente de sua estrutura e complexidade, através de um objeto daquela base com as suas respectivas operações . A O.O. concentra seus esforços na busca de modularidade, simplicidade e na capacidade de evolução de código, a fim de possibilitar que novos requisitos possam ser incorporados ao sistema, sem levar em conta detalhes de implementação. Procura também diminuir o "gap" semântico entre o mundo real e as soluções projetadas, empregando-se níveis distintos de abstração, que vão

refinando-se sucessivamente, partindo sempre do mundo real para o das soluções e nunca ao contrário. Os SGBDOOs dentro da tendência mundial de integração de ambientes, transforma-se em componente importante no âmbito computacional, já que combinam as funcionalidades dos SGBDs com os conceitos de O.O., tornando-se o repositório ideal para as informações que são compartilhadas por múltiplos usuários, mútiplos produtos e em diferentes plataformas. Inúmeros SGBDOOs foram construídos nos últimos anos, com vistas a esses objetivos, tais como : ORION (BANE87), IRIS (WILK90), O_2 (BANC89a), GemStone (PENN87), POSTGRES (STON90), etc. Entretanto, não existe ainda um modelo de dados único em O.O., apresentando cada um desses sistemas o seu próprio modelo de dados.

Se a tarefa de projetar BDs já era difícil para os usuários nos modelos tradicionais, o problema vem se agravar ainda mais com esse novo direcionamento, haja visto o aumento de complexidade das novas aplicações e dos conceitos embutidos no paradigma da O.O.

Neste artigo é descrito um modelo de BDOO, bem como a implementação de uma ferramenta que possibilite, de forma amigável, a aquisição e a manutenção do esquema do BD que mapeie o modelo em questão. Este módulo é componente de um ambiente integrado do SGBDOO em desenvolvimento no IME. O presente artigo é estruturado da seguinte forma : em II são levantados os principais conceitos de O.O.; em III é feita uma descrição do modelo conceitual MORSO (Modelo em Rede Semântica para Objetos); em IV é apresentado o desenvolvimento da ferramenta de aquisição e manutenção de esquema, suas características e detalhes importantes de implementação; em V é feita uma conclusão geral sobre o andamento do projeto ora em estudo.

II - PRINCIPAIS CONCEITOS DA ORIENTAÇÃO A OBJETOS

Neste parágrafo descreveremos os principais conceitos da O.O, cujas características devem estar presentes em todo sistema

classificado segundo esse paradigma. São eles:

A - TIPO ABSTRATO DE DADOS (TAD): Abstração de dados é o mecanismo pelo qual modela-se ao mesmo tempo a estrutura de dados e as operações sobre ela. Assim, os objetos consistem de uma interface externa e uma implementação interna, que é encapsulada. A interface é constituída de um conjunto de operações que se aplicam sobre o objeto, formando a sua parte visível. A implementação possui a representação dos dados e das operações permitidas sobre eles, isto é, o código que implementa a operação

O objetivo de se esconder os dados é permitir uma certa independência entre o comportamento do objeto e a sua implementação. As principais vantagens do TAD são : permite uma melhor modelagem do mundo real, separa a implementação da especificação e permite que sistemas sejam extensíveis.

B - CLASSE : Classe é o construtor mais comumente usado para definir tipos abstratos de dados nas linguagens de programação orientada a objetos. A Classe incorpora a definição da estrutura e as operações permitidas sobre os elementos pertencentes àquela classe. Assim, através de uma classe define-se um TAD (KHOS90). Entretanto, a classe é mais que um TAD, pois ela incorpora dois aspectos importantes tais como: uma fábrica de objetos, ou seja, gera novas instâncias de objetos, e de um depósito de objetos, isto é, armazena todos os objetos de uma classe. A definição de uma classe é formada também por um conjunto de métodos, que definem o seu comportamento. Os métodos nada mais são do que um conjunto de funções que executam alguma operação sobre as variáveis de instância dos objetos. Para se ativar um método, deve-se enviar uma mensagem ao objeto.

C - SOBRECARREGAMENTO E "BINDING" DINÂMICO : Sobrecarregamento é a facilidade que permite definir operações de mesmo nome em classes diferentes, porém com semântica e implementação distintas. O

sobregarregamento está ligado ao "binding" dinâmico, significando que o sistema associará o seletor da mensagem ao método que o implementa em tempo de execução.

D - Herança : A idéia essencial da herança é fornecer um mecanismo simples e poderoso para definir novas classes, de forma que estas herdem as propriedades das classes existentes. Na maioria das linguagens orientadas a objetos, uma classe herda tanto os métodos, aspecto comportamental, como as variáveis de instância, aspecto estrutural. A herança fornece um bom mecanismo para a organização da informação, entretanto a sua mais importante contribuição é o compartilhamento ou reutilização de código. Se uma classe só possuir uma única superclasse imediata esta herança é denominada de herança simples , cuja hierarquia de classes forma uma árvore; no caso de mais de uma superclasse imediata, a herança é chamada de herança múltipla , e a hierarquia de classes é estruturada sob forma de um grafo acíclico direcionado (GAD).

Como as propriedades (variáveis de instância e métodos) de uma classe são formadas pela união das propriedades de todas as suas superclasses e das propriedades definidas na própria classe, pode ocorrer que classes antecessoras possuam propriedades com o mesmo nome, mas com semântica totalmente diversa. Quando a classe tenta herdar a mesma propriedade de superclasses distintas, surge o que é denominado de conflito de herança.O principal problema da herança múltipla são as estratégias para a resolução deste conflito, estudadas em (ROSS91).

E - IDENTIDADE DO OBJETO : Num sistema plenamente O.O., a cada objeto deve ser associado uma identidade a qual permanece como identificador, independentemente da sua estrutura ou do seu estado. É um conceito interno do objeto, cujo propósito é fornecer um meio de representar a sua individualidade, independentemente de como é acessado, do seu conteúdo e onde se encontra. A técnica

mais poderosa para suportar identidade é através de "surrogates", cujo identificador único e global é gerado automaticamente para cada objeto, de forma que este seja completamente independente do seu estado e endereço do objeto.

Bancilhon (BANC89b) propõe que os SGBDOOs devam satisfazer a dois critérios : incorporar os principais conceitos da O.O. e possuir as facilidades dos SGBDs "tradicionais", a exemplo de (KHOS90) : persistência, controle de transações e de concorrência, recuperação, linguagem de consulta, restrições de integridade, segurança, desempenho e versionamento. Os SGBDOOs buscam ainda a completude computacional, significando que qualquer função computável deve poder ser programada através do sistema.

III - MORSO - MODELO EM REDE SEMÂNTICA PARA OBJETOS

O MORSO (XAVI89) (XAMO90) é um modelo conceitual O.O, cujo objetivo é capturar os objetos do mundo real, utilizando-se de uma Rede Semântica, extensão do modelo MORSE (BOUZ86), para a representação do conhecimento. O MORSO consiste de uma dupla onde RS é a Rede Semântica que permite modelar conceitualmente a aplicação, e LC é a Linguagem Conceitual que permite manipular a rede, definindo e alterando o esquema conceitual do modelo. A Rede Semântica possui uma representação gráfica, através de um grafo acíclico orientado consistindo de nós e arcos, e uma representação interna, proposta para ser executada através de uma Base de Conhecimento (Base de Fatos e Regras), o que confere à rede um mecanismo de dedução. A modelagem dos objetos emprega os conceitos básicos de orientação a objetos. A figura (1) ilustra a representação gráfica de um objeto para o MORSO.

A rede MORSO, formalizada em (XAVI89), é definida pela quádrupla :**MORSO ::= < CN, CA, CR, CO >** onde :CN : Categoria de Nós; CA : Categoria de Arcos; CR : Categoria de Restrições de

Integridade; CO : Categoria de Operações em Tipos Básicos.

As categorias serão resumidamente descritas a seguir :

A) Categoria de Nos (CN) : os nós representam os objetos modelados pela rede. A rede admite duas categorias de nós : classe de objetos (cl), que representa a descrição das classes dos objetos, e instância de objeto (io), que representa a ocorrência de um determinado objeto.

B) Categoria de Arcos (CA) : os arcos são responsáveis pela interligação dos diversos nós, indicando por meio da categoria e do sentido, a classificação dos objetos que estão sendo ligados. Os principias arcos são : agregação (a), servindo como construtor de tuplas; generalização (g), indicando uma relacionamento "É-UM"; conjunto (s) e lista (l), utilizados para construir conjuntos e listas respectivamente.

C) Categoria de Restrições (CR) : as restrições representam uma informação adicional acerca da classe ou da instância do objeto. O MORSO classifica as restrições em : restrições associadas ao valor do objeto atômico e restrições nas associações entre objetos.

D) Categoria de Operação (CO) : determina as operações permitidas sobre os tipos básicos, tais como : projeção, seleção e junção para tuplas, união e interseção para conjuntos, etc.

Com o objetivo de fornecer maior flexibilidade à rede, no que diz respeito a sua implementação, algumas alterações foram acrescidas ao MORSO (ROSS91) :

A - CLASSES : o modelo é constituído de objetos. Todo objeto possue uma descrição de seu estado e das operações que ele pode realizar, denominada classe. Neste trabalho, a exemplo do ORION (KIM90a), consideramos a classe como sendo formada por uma agregação de variáveis locais, chamadas no modelo de variáveis de instância, associadas a domínios que tanto podem ser outras lasses como tipos primitivos definidos no sistema. A forma do

objeto passa a ser descrita pelas variáveis de instância e o comportamento, pelos métodos. As classes não são dependentes de outras classes, possuindo existência própria. Uma classe pode ser referenciada por qualquer número de classes. A ligação entre uma classe e suas classes componentes passa a ser feita através de suas variáveis de instância. Esta ligação está relacionada ao compartilhamento ou não de objetos que é dependente da semântica das classes componentes, e pode ser classificada em : ligação por referência e ligação por dependência.

1. **Ligação por referência** : o objeto referenciado através de uma ligação por referência não possui um vínculo de propriedade com o objeto que o referencia, isto é, este objeto pode ser referenciado por outros objetos (compartilhamento de objetos).

2. **Ligação por dependência** : esta ligação representa uma composição, isto é, o objeto é composto de uma agregação de outros objetos, existindo uma relação de propriedade exclusiva entre as partes do objeto. Este tipo de ligação está presente no ORION com o nome de "ligação composta", sendo utilizado para a criação dos objetos compostos. A ligação por referência e a ligação por dependência são a base para a construção dos objetos compostos. Existe, no entanto, a necessidade de se propiciar ao usuário um mecanismo que permita a especificação de "classes dependentes", isto é, classes cuja existência sejam dependentes da existência de outra classe. Adotou-se a solução da criação de valores complexos, semelhante ao encontrado no O_2. A vantagem desta solução está na simplicidade de definição, pois o usuário não precisa definir uma nova classe quando na verdade deseja definir apenas um valor composto, visto que não haverá qualquer tipo de compartilhamento, nem da sua definição nem das suas instâncias. A definição destes valores passa a fazer parte da definição da classe principal.

B - **OBJETO COMPOSTO** : o paradigma da O.O., na sua forma

tradicional, é suficiente para representar uma coleção de objetos relacionados. A hierarquia de classes captura o relacionamento "É-UM" entre uma classe e suas subclasses. Entretanto, a hierarquia de classes não captura o relacionamento "É-PARTE-DE" entre um objeto e os objetos por ele referenciados. Este relacionamento é representado pelo conceito de objeto composto. Objeto composto é uma coleção de instâncias relacionadas, formando uma estrutura hierárquica de estrutura bidimensional (KIM90b). O objeto composto introduz uma restrição de integridade no modelo O.O. através da noção de objeto dependente. Um objeto dependente é um objeto que depende da existência de um outro objeto. Com isto, a instanciação de um objeto composto deve ser realizada de cima para baixo, ou seja, o objeto raiz de uma hierarquia de objeto composto deve ser criado primeiro.

O MORSO apresenta uma definição para a semântica de evolução do esquema. Esta evolução é composta de uma conjunto de propriedades do esquema denominadas de "invariantes", e de um conjunto de regras que guiam a seleção do melhor caminho para a preservação dessas mesmas propriedades, além das restrições de implementação denominadas de "diretivas" de implementação (ROSS91). No MORSO as mudanças estão divididas em três categorias:

1. Alteração do conteúdo de uma classe : correspondem basicamente às operações de adição e exclusão das variáveis de instância, métodos e restrições das classes.
2. Alteração de um arco : corresponde à inclusão e exclusão de um arco de generalização entre duas classes.
3. Alteração de uma classe : operação que permite a exclusäo e inclusäo de classes no esquema.

IV - MAME : MÓDULO DE AQUISIÇÃO E MANIPULAÇÃO DE ESQUEMA

Inicialmente o modelo MORSO foi desenvolvido com planos para

ser construido em PROLOG, razão pela qual cada um dos seus conceitos (arcos,nós,restrições, etc) foi estruturado sob forma de claúsulas de Horn. Ao se dar início a fase de desenvolvimento do módulo, verificamos que uma série de pré-requisitos considerados de relevada importância para o projeto (ROSS91) não seriam satisfatórios em PROLOG, fazendo-nos direcionar a implementação do sistema para a linguagem C. Com esta nova abordagem, as representações das estruturas básicas do modelo anteriormente previstas sofreram alterações, sendo apresentadas a seguir:

A - OBJETO : Um objeto na rede MORSO é definido como tendo a seguinte forma :

```
ioC"Identificador","Versão","Nome_Classe","Valor").
```

A representação proposta neste trabalho é mostrada na figura (2.a) onde :

1. Identificador : gerado pelo sistema, é baseado no conceito de "surrogates", a partir da data e hora de criação da classe .

2. Tamanho : tamanho em bytes do objeto. Como os objetos possuem tamanho variável, este campo é utilizado pelo gerenciador de objetos para a sua manipulação.

3. Identificador da Classe : como todo objeto é uma instância de uma classe, o sistema deve possuir um modo de identificar a que classe ele pertence.

4. Versão : apesar deste trabalho não ter abordado o versionamento de objetos, ele já está previsto no modelo, possibilitando o seu desenvolvimento posterior.

5. Valores : Corresponde aos valores assumidos pelos diversos componentes do objeto. A determinação exata se os valores reais dos componentes dos objetos ou seu identificador serão armazenados é uma decisão a ser tomada pelo módulo responsável pelo armazenamento dos objetos. Nesse módulo serão utilizadas técnicas de agrupamento entre objetos ("cluster") relativas as suas

superclasses e componentes. Este trabalho, bem como os métodos de acesso a esses objetos, está em desenvolvimento (FREI91).

B - CLASSE : Uma classe foi definida no MORSO como tendo a seguinte forma geral :

"cl("nome_classe","nome_tipo",["lista de operadores"]).

Considere o exemplo:

cl(aluno, tupla, [matricula, dar_nota, formar]).

O conceito inicial propunha que a classe possuisse apenas o seu nome e tipo. Logo de inicio surgiu o seguinte problema : considere no exemplo anterior a definição da classe "aluno". Analisando-se somente a definição, não encontramos os campos correspondentes às tuplas de aluno. Esta informação, pelo modelo, está armazenada nas instâncias de aluno. Esta modelagem exige que, ao se analisar uma classe busquemos pelo menos uma de suas instâncias e, portanto, na definição de uma classe deve-se incluir pelo menos uma instância desta classe.

Outro conceito introduzido pelo MORSO é o de classe componente, isto é, uma classe é formada pela agregação de classes componentes. Neste trabalho foi proposto que uma classe seja formada pela agregação de variáveis de instância associadas a domínios, que tanto podem ser outras classes como tipos primitivos definidos no sistema. A forma geral para a definição de uma classe é ilustrada na figura (2.b), onde :

1. Identificador : identificador gerado pelo sistema, de forma análoga ao gerado para o objeto.

2. Versão : como já mencionado anteriormente, não foi implementado na presente versão.

3. Lista de Superclasses : o modelo proposto, ao contrário do Smalltalk, aceita herança múltipla necessitando para isto, de uma lista ordenada, sem repetição, das superclasses. A implementação proposta é feita através de uma lista encadeada.

4. Lista de Subclasses : a exemplo da lista de superclasses, esta lista contem as subclasses imediatas da classe. O objetivo desta lista é facilitar o caminhamento da consulta.

5.Lista de Restrições : o MORSO propõe uma série de restrições para uma classe, denominadas de "restrições associadas aos arcos de generalização e especialização". Esta restrição associa uma determinada condição a uma variável de instância herdada a partir de uma classe mais genérica.

6. Lista de Campos : como já definido, uma classe é composta de uma coleção de variáveis de instância. A lista de campos é uma coleção de variáveis de instância no seguinte formato :

nome	tipo	tamanho	característica	lista restrições

onde :

1) nome : nome da variável de instância, semelhante ao nome do atributo no modelo relacional. Possui escopo local a classe, isto é, mais de uma classe pode possuir o mesmo valor para nome.

2) tipo : corresponde a identificação do tipo de domínio ao qual esta variável é associada. No modelo proposto pode-se ter os seguintes casos :

- tipo primitivo: inteiro, real, data, longo, etc ;
- valor composto: formado a partir dos construtores Tupla,Lista ou Conjunto;
- classe definida pelo usuário.

Com o objetivo de permitir a manipulação de objetos "multimídia", é previsto o tipo Longo, podendo suportar Texto, Som e Imagem.

3) tamanho : define o tamanho do campo em bytes.

4) característica : este campo informa duas características da variável de instância : unicidade e valores nulos para valores atômicos. Este campo também é usado para o sistema identificar se a ligação é por dependência ou por referência.

5) lista de restrições: estas restrições referem-se as propostas para os atributos atômicos no MORSO : restrições de domínio (valor e intervalo) e restrições de estado.

C - **RESTRIÇÕES** : As restrições no MORSO não são especificadas separadamente de sua estrutura, fazendo parte integrante da definição do objeto. As restrições reproduzem informações adicionais a respeito das classes ou das suas instâncias. Nos modelos de BDOO, não é comum encontrá-las explicitamente definidas, pois o paradigma de O.O. possibilita a inclusão das restrições através dos métodos. O Morso define uma série de restrições, sendo as seguintes implementadas no protótipo atual :

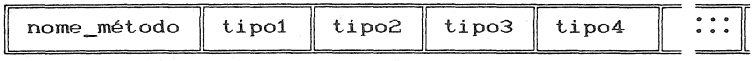
1. Restrições de "domínio" : as restrições de domínio referem-se somente aos objetos atômicos definidos no modelo. As restrições de domínio implementadas são : restrição de valor e restrição de estado;

2. Restrições nas associações entre objetos : são definidas as restrições associadas ao arco de agregação (restrição de unicidade e de existência), a restrição de cardinalidade e a de participação de uma instância numa classe.

As restrições foram implementadas como uma agregação de dois campos: tipo e valor. O campo tipo é constituido de um código numérico indicando qual o tipo de restrição escolhida e o campo valor foi implementado como uma sequência de caracteres.

D - **MÉTODOS** : no paradigma de O.O., os objetos são encapsulados, isto é, os seus valores internos só podem ser acessados através de rotinas específicas definidas para este fim, denominadas de métodos. Os objetos no MORSO são passivos, segundo a classificação de Takahashi (TAKA89), isto é, necessitam que uma mensagem lhes seja enviada para que se tornem ativos. As mensagens ativam os métodos que executam as operações sobre os objetos. O usuário, à semelhança do O_z , deve especificar uma "assinatura" para os

métodos. Esta "assinatura" ou protocolo é utilizada para a verificação de tipos no momento da ativação do método. A representação do protocolo de um método é dada por :



onde : **tipo1** : tipo do retorno do método;
tipo2, tipo3, etc : tipo dos parâmetros.

Assim como no Smalltalk, o sistema foi desenvolvido supondo-se que o esquema do banco de objetos, isto é, as descrições das classes definidas pelo usuário, estejam na memória "RAM" do micro. Esta obrigatoriedade está diretamente relacionada à eficiência do sistema. A descrição de uma classe está ligada diretamente às suas superclasses. Portanto para se acessar uma variável de instância ou um método é necessário, às vezes, acessar toda a hierarquia desta classe. Para facilitar este acesso às descrições das classes, elas foram organizadas num vetor ("array"), alocado dinamicamente durante o início da carga do sistema, e denominado Diretório, cuja estrutura é ilustrada na figura (2.c), onde :

1. Nome_Classe: nome fornecido pelo usuário no momento de criação da classe. Na criação de um valor composto, o sistema preenche automaticamente com o nome do tipo do construtor escolhido pelo usuário (TUPLA,LISTA ou CONJUNTO).

2. Identificador: o mesmo que é fornecido para o descritor da classe. Esta duplicação facilita o acesso, pois as rotinas internas acessam o diretório via identificador.

3. Número_Arquivo: campo deixado para ser preenchido pelo módulo de armazenamento de objeto (FREI91).

4. Lista_SuperClasse :lista de todas as superclasses de uma determinada classe. Esta lista é utilizada para a instanciação dos objetos (FREI91).

5. Lista_Atributos: visando facilitar as rotinas de

atualização, criou-se uma lista onde ficam preservados os pares: nome da variável de instância e nome da classe de origem de todas as variáveis de instância herdadas por uma classe. Com esta lista, o sistema possui todas as informações necessárias a propagação e resolução dos conflitos de herança que possam surgir. Evita-se a construção de rotinas recursivas, pois só será necessário acessar as superclasses imediatas de cada classe, no caso de conflito de herança.

6. Lista_Métodos : uma lista de métodos herdados à semelhança da Lista_Atributos.

7. Descritor_Classe: é um ponteiro para a estrutura da classe, descrita anteriormente.

A figura 3 ilustra um exemplo da representação das classes Funcionário e Departamento no MAME.

V - CONCLUSÃO

O paradigma de orientação a objetos pode ser definido através da seguinte expressão (KHOS90) :

Orientação Objeto =

Tipo Abstrato de Dados + Herança + Identidade do Objeto

Da necessidade de combinar as propriedades do paradigma de Orientação a Objetos com as funcionalidades dos SGBDs surgiram os SGBDOOs, como sendo (KHOS90):

SGBDOO = Orientação a Objetos + Funcionalidades dos SGBDs

O MAME é parte integrante de um SGBDOO ora em desenvolvimento no IME. Apresenta um interface amigável para aquisição de objetos, atendendo as expectativas iniciais previstas pelo projeto. Deverá ser integrado brevemente ao gerenciador de objetos, responsável pelo acesso, armazenamento e manipulação dos mesmos, e ao módulo de consulta, cuja linguagem encontra-se em fase de especificação.

Os estudos sobre Bancos de Dados Orientado a Objetos

encontram-se ainda em fase de padronização. Os anos 90 despontam como a década em que os SBGBDOOs serão exaustivamente pesquisados em busca de padronização e consolidação desta nova tecnologia.

Referências Bibliográficas

(ALBANO85) -ALBANO, A.; CARDELLI, L; ORSINI, R. "**Galileo: A Strongly-Typed Interactive Conceptual Language**". ACM Transactions on Database Systems, 10(2), June, 1985.

(BANC89a) - BANCILHON,F. et alii. "**A Query Language for the O_2 Object-Oriented Database**", in Proc. Second Workshop Database Programming Languages, 1989.

(BANC89b) - BANCILHON, F. . "**Query Languages for Object-Oriented Database Systems : Analysis and a Proposal**". 4º Simpósio Brasileiro de Banco de Dados, Campinas - S.P., abril de 1989.

(BANE87)- BANERJEE,Jay et alii. "**Data Model Issues for Object-Oriented Aplications**".ACM Transaction on Office Information Systems, Vol. 5, No 1, Janeiro 1987.

(BOUZ86) - BOUZEGHOUB, B. . "**Un Systeme Expert en Conception de Systemes D'Informations**". Dissertação de Doutorado. L'Université de Paris VI, 1986.

(CHEN76) - CHEN, P. . "**The Entity Relationship Model - Toward a Unified View of Data**". ACM TODS, Vol. 1, Num. 1, 1976.

(CODD70) - CODD, E. . "**Relational Model of Data for Large Shared Data Bank**". ACM TODS, Vol 13, Num. 6, 1970.

(DITT86) - DITTRICH, K.R. ."**Object-Oriented Database Systems**". Proc. 5th ER Conference 1986, North Holland, 1987.

(FREI91) - FREITAS, L. O. ."**Esquema Físico para Armazenamento de Objetos Complexos**". Tese de Mestrado, IME, (em desenvolvimento).

(KHOS90) - KHOSHAFIAN, S & ABNOUS, R. "**Object Orientation Concepts, Languages, Databases, User Interfaces**". John Wiley & Sons,Inc.,1990.

(KIM90a) - KIM, W. et alii. "Architecture of the ORION

Next-Generation Database System". IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No 1, Março 1990.

(KIM90b) - KIM, WON. "**Object-Oriented Databases: Definition and Research Directions**". IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No 3, Setembro, 1990.

(MYLO82) - MYLOPOULOS, J.; BERNSTEIN, P.; WONG,H.A. . "**Language Facility for Designing Database Aplication**". ACM TODS, Vol. 5, Num. 2, 1982.

(PENN87) - PENNEY, J. & Stein J. ."**Class Modification in the GemStone Object-Oriented DBMS**". OOPSLA, 1987.

(ROSS91) - ROSSETTI, L.C. "**Linguagem de Definição e Manipulação de Esquema num Banco de Dados Orientado a Objetos**". Tese de Mestrado, IME , Rio de Janeiro, BR, fevereiro,1991.

(STON90) - STONEBRAKER, M. et alii. "**The Implementation of POSTGRES**". IEEE Transactions on Knowledge and Data Engineering. Vol.2, No. 1, March 1990.

(SU83) - SU,S. Y. ."**SAM*** : **A Semantic Association Model For Corporate**". Inf. Science, Num. 29, 1983.

(TAKA89) - TAKAHASHI, Tadao. "**O Paradigma de Objetos: Introdução e Tendencias**".VIII Jornada de Atualização em Informática, Urberlândia, 1989.

(XAMO90) - XAVIER, C.M.S. & MOURA, A.M.C."**Modelagem Conceitual Orientada a Objetos para Aplicações Não Convencionais em Banco de Dados**". 5º Simposio Brasileiro de Banco de Dados, Rio de Janeiro - RJ, abril,1990

(XAVI89) - XAVIER, Carlos Magno da Silva. "**Modelagem Conceitual Orientada a Objetos para Banco de Dados**". Tese de Mestrado, IME, dezembro,1989.

(WILK90) - WILKINSON, K. et alii. "**The Iris Architecture and Implementation**". IEEE Transactions on Knowledge and Data Engineering, Vol.2, No.1, March 1990.

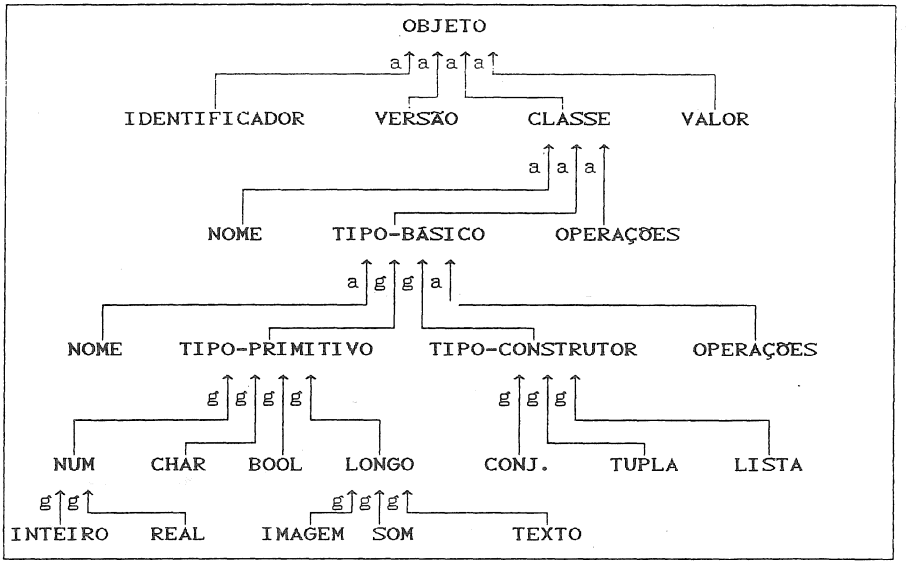


FIGURA 1: Representação gráfica de um objeto no MORSO.

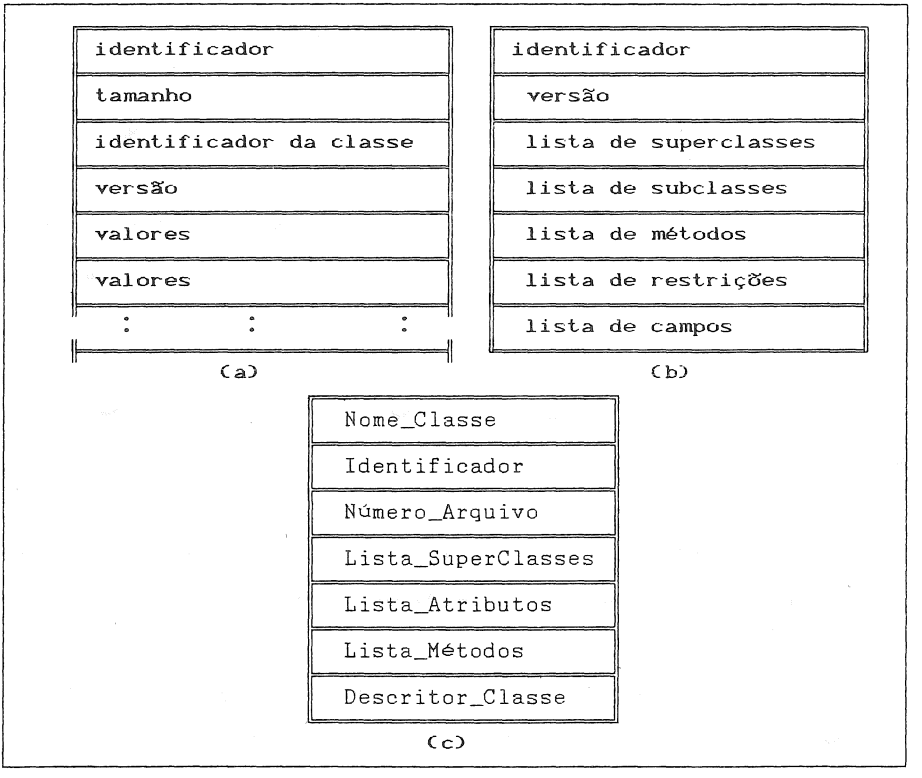


FIGURA 2: Representação das estruturas internas do MAME

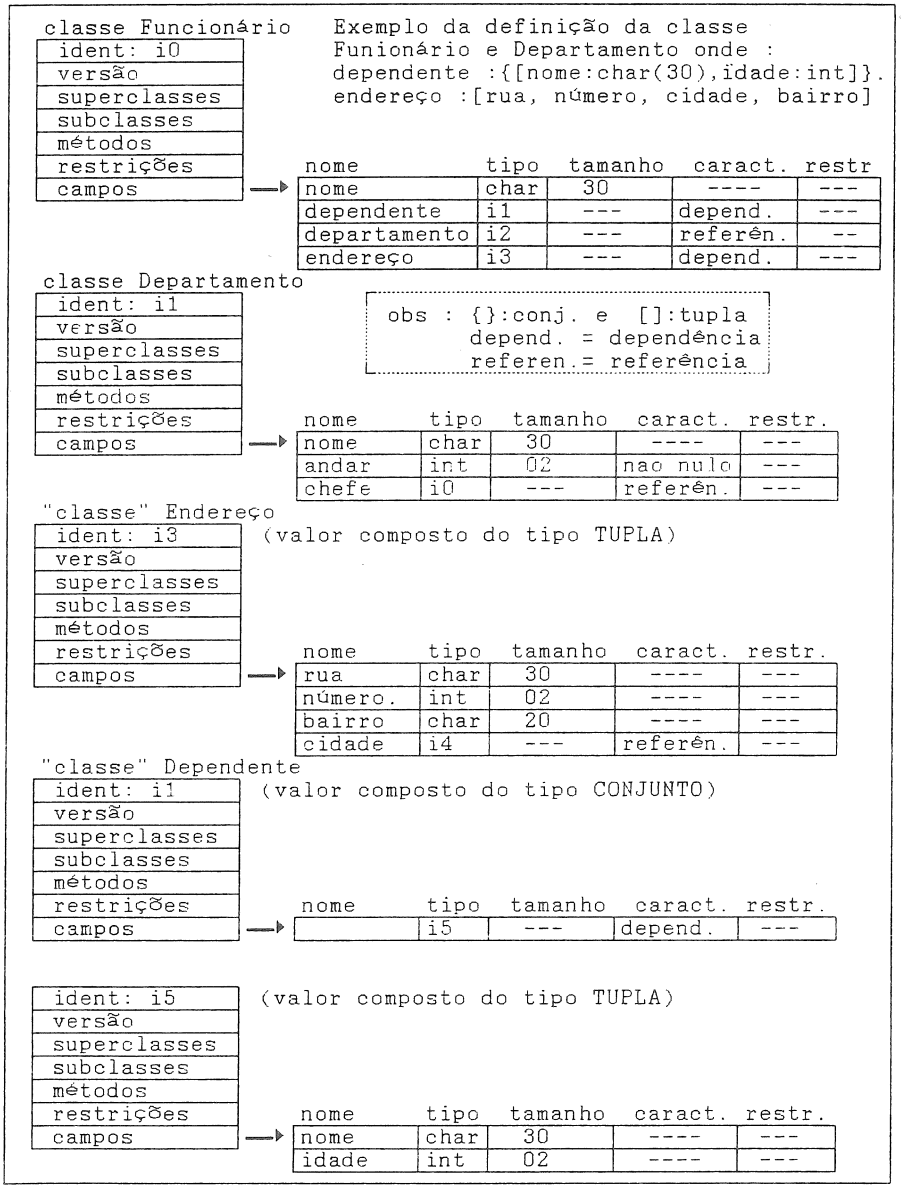


FIGURA 3: Exemplo de representação de classes no MAME

ROBCAD
C.A.D. para Robótica

Osvaldo Alberto Driollet
Universidad Nacional de San Juan
Av. Libertador Gral. San Martín 1109 (Oeste)
5400 - San Juan Argentina

RESUMEN

ROBCAD es un C.A.D. para robótica diseñado para asistir al estudio de las regiones de acceso de un manipulador robótico y a su problema cinemático, considerando este último tanto en sus aspectos directo como inverso.

En ambas áreas permite la solución de las siete estructuras de manipuladores robóticas más importantes: rotacional, cuadrilátero deformable, S.C.A.R.A., polar, tipo Puma, cilíndrica y cartesiana. El estudio de las regiones de acceso permite graficación en alta resolución en 2D y 3D, conocimiento, comparación y superposición de áreas y volúmenes de acceso, determinación de distancias máximas, mínimas y entre puntos, programación de tares en forma gráfica en 2D y 3D, así como otros tópicos relacionados a esta área.

Soluciona la cinemática directa no sólo de las estructuras predefinidas arriba nombradas, sino también la de cualquier estructura de diseño del usuario. Posibilita la obtención de cualquiera de las matrices de transformación de la estructura y el conocimiento de los vectores normal y orientación de cualesquiera de los elementos de la misma. Permite la generación autoregresiva de las expresiones cinemáticas simbólicas del manipulador.

Permite solucionar la cinemática inversa mediante trigonome-

tría espacial, posibilitando obtener soluciones numéricas y gráficas en 2D y 3D en una amplia variedad de aspectos.

1. Introducción

El presente trabajo intenta describir la potencialidad ofrecida por ROBCAD al usuario que aborda el problema de la solución cinemática de un manipulador robótico, y de su región de acceso, considerando el presente estudio para manipuladores de hasta seis grados de libertad.

El estudio de las regiones de acceso de un manipulador es un importante tópico pocas veces abordado, y permite mediante su conocimiento geométrico y analítico, la determinación de las reales bondades operativas del mismo.

Esto es, permite conocer de una forma completa las posibilidades de acceso y planificación de tareas dentro del ámbito de trabajo definido por el tipo particular de estructura del manipulador considerado.

Por orto lado la cinemática del brazo del robot trata sobre el estudio analítico de la geometría de movimiento del mismo con respecto a un sistema coordenado de referencia fijo en función del tiempo, sin considerar las fuerzas o momentos que originan dichos movimientos. En este sentido pueden definirse dos planteos:

1) Para un manipulador determinado, dado el vector de coordenadas generalizadas q(t), (rotaciones y traslaciones de cada juntura), y los parámetros geométricos de la estructura, con n grados de libertad, determinar cual es la orientación y la posición del extremo final del manipulador con respecto a un sistema de coordenadas de referencia. Este constituye el problema Cinemático Directo.

2) Dada la posición y orientación deseada del elemento final del manipulador y los parámetros geométricos de la citada estructura con referencia a un sistema fijo, saber si puede el manipulador alcanzar la posición y orientación de la mano que se desea. Y también, si es posible, determinar cuántas configuraciones diferentes del manipulador satisfacerán la misma condición. Esta cuestión cosntituye el problema Cinemático Inverso.

2. Motivaciones

Las motivaciones fundamentales del trabajo están dadas en la necesidad de establecer una relación entre el espacio articular y el espacio cartesiano, y su relación inversa, mediante un ordenador personal, por cuanto la complejidad matemática de los cálculos requeridos, así como la diversidad de estructuras y parámetros de diseño involucrados hacen que el proyecto de un paquete de software con tales características resulte de sumo interés al diseñador. Este enfoque se complementa con similares razones relacionadas a las regiones de acceso, posibilitándose un estudio total tanto de las facilidades operativas de un manipuladror relacionadas a su ámbito de trabajo, así como de sus particularidades cinemáticas.

Este tipo de herramientas computacionales muy utlizadas en diferentes ramas de la ingeniería se conoce como diseño asistido por computador (C.A.D.).

ROBCAD se propone de manera tal que se pueda facilitar el diseño y/o estudio de distintas estructuras de manipuladores robóticos con un mínimo de tiempo y esfuerzo, dando la posibilidad al usuario de elegir entre un amplio menú de opciones de manera de poder conocer su diseño de una forma acabada.

3. Estructura informática

Para lograr un sistema de mayor versatilidad y mejor funcionalidad se definió una estructura en base a overlays o recubrimientos. Estos recubrimientos han sido ubicados en el sistema en las discontinuidades lógicas del código, como son las áreas referentes a los menúes de opciones brindados al usuario, posibilitando así optimizar el rendimiento en función del tiempo.

Asimismo la comunicación entre módulos se efectuó en base a la definición de variables globales permitiendo con la citada estructura obviar la grabación y lectura de archivos de comunicación y transferencia de variables, logrando el consiguiente ahorro de tiempo de acceso a disco y de espacio físico.

Por otra parte todo el sistema se diseñó con una técnica modular, para permitir futuras ampliaciones de las posibilidades del mismo. Así, un futuro módulo informático puede ser incorporado sin ningún tipo de problemas, aumentando de este modo la potencia de todo el paquete disponible.

4. Descripción general de las posibilidades de ROECAD

4.1 Descripción preliminar

Todo el sistema ha sido desarrollado pensando en atender no sólo a una elevada confiabilidad y rapidez, sino también en hacerlo amigable al usuario, logrando combinar calidad visual y eficacia. Es por esta razón que toda la presentación en pantalla, aclaraciones, comentarios, advertencias, ayudas, gráficos en alta resolución y hasta los mismos menúes de opciones se efectúan en ventanas colocadas a tal efecto en pantalla. Se busca de este modo no sólo la exac-

titud de los resultados sino también la calidad en general de todo el sistema.

4.2 Descripción particular del módulo de regiones de acceso

4.2-1 Conceptos previos

Una de las razones de mayor importancia en la elección de un tipo particular de geometría de un manipulador robótico viene dada por ámbito o volumen de trabajo. Esta característica es de suma importancia porque permite establecer las ventajas operativas que una estructura de manipulador presenta de acuerdo a su geometría. Debido además a que toda planificación de tareas se hace obviamente dentro del ambiente de trabajo alcanzable por el manipulador, el conocimiento analítico, gráfico y geométrico del mismo se vuelve de singular importancia. Si a este hecho se le suma la posibilidad de interelacionar las bondades ofrecidas por distintos tipos de geometrías de manipuladores entre sí, entonces la necesidad de una herramienta computacional de diseño se hace más justificable.

4.2-2 Posibilidades ofrecidas

El módulo de Robcad que aborda el problema y solución al estudio de las regiones de acceso fue diseñado para trabajar en forma aislada o bien en forma conjunta con el módulo de estudio cinemático directo e inverso.

Robcad posibilita el estudio analítico de los siete tipos de estructuras predefinidos en este módulo, que como se mencionó anteriormente son:

estructura rotacional

estructura cuadrilátero deformable

estructura S.C.A.R.A.

estructura polar

estructura cilíndrica

estructura tipo Puma

estructura cartesiana

Para todas las estructuras mencionas permite el análisis que a continuación se detalla:

- Permite una vez ingresado el dimensionamiento geométrico del manipulador conocer la superficie de las vistas lateral y superior de la región de acceso, así como su volumen total.
- Posibilita las ayudas al usuario en alta resolución en gráficos bi y tridimensionalse graficando las distintas opciones de ingreso de acuerdo al tipo de sistema de referencia elegido para un manipulador en particular.
- Obtiene las vistas lateral y superior de la región de acceso en dos y tres dimensiones, así como el volumen de trabajo del manipulador en 3D. Todos los gráficos ofrecidos por el sistema usando sistema de representación en 3D, brindan esta alternativa trabajando sobre algoritmos de proyección central o cónica, los que permiten definir el gráfico completamente a gusto del usuario. Este puede indicar distancia de observación, escala del dibujo, rotación del solido respecto a los tres ejes cartesianos principales o respecto a un eje cualquiera, traslación del sólido respecto al mismo sistema, tipo de línea e intensdad del dibujo, etc. De esta forma se permite en todo el estudio 3D hasta agotar todas las inquietudes del ususario del sistema.
- Los gráficos mencionados en el punto anterior son completamente compatibles en línea con la posibilidades en 2D, pudiendo tenerse

conjunciones de ambos tipos en un momento dado.

- Permite la determinación de coordenadas de puntos dentro de un área o volumen de trabajo, la determinación en forma analítica y gráfica (2D y 3D) de distancias máximas, mínimas o entre puntos cualesquiera.
- Posibilita la compración y superposición de datos en forma analítica y gráfica (2D y 3D) lo que permite un estudio de interrelación entre distintos tipos de manipuladores, ya sean de igual o distinta geometría.
- Este módulo permite además la simulación de trayectorias en el plano o en el espacio en perspectiva 3D, sea cual fuere la tarea que se le asigne al manipulador dentro del espacio de trabajo del mismo. Esta simulación se realiza incluso considerando al manipulador dotado de una mano de tres grados de libertad. Esta posibilidad le permite al usuario estudiar posteriormente, por separado, las trayectorias generadas, permitiendo un estudio gráfico completo de las mismas.

Estos son, descriptivamente, los aspectos sobresalientes del módulo desarrollado para el estudio de las regiones de acceso.

4.3 Descripción particular de la solución al Problema Cinemático Directo

4.3-1 Consideraciones teóricas

El problema cinemático directo se reduce a encontrar una matriz de transformación que relaciona el sistema de coordenadas ligado al cuerpo con el sistema de coordenadas de referencia. La representación espacial de un elemento mecánico rígido en forma matricial, que incluye los movimientos rotacionales y traslacionales del sistema ligado al cuerpo, y que describe la geometría espacial de

un brazo fue desarrollada por Denavit y Hatemberg [1955]. El paquete aquí descripto utiliza esta representación dada su universalidad para derivar las ecuaciones cinemáticas directas de un brazo.

4.3-2 Posibilidades ofrecidas

ROBCAD soluciona el problema cinemático directo, permitiendo al usuario atender todos los aspectos de su solución, desde el cálculo numérico al cálculo simbólico pasando por la graficación del manipulador hasta la impresión de resultados.

Se detallan seguidamente las posibilidades ofrecidas que permiten entre otras cosas desarrollar las siguientes tareas:

- Consultar ayuda aclaratoria sobre el concepto de solución cinemática directa y sobre las posibilidades ofrecidas por ROBCAD.
- Consultar ayuda sobre el método de cálculo utilizado y sobre la forma de determinar las variables articulares y parámetros geométricos de la estructura.
- Ingresar los parámetros y variables articulares del manipulador, dando la opción al usuario de modificar y/o confirmar los mismos cuantas veces lo desee, antes de optar por realizar el cálculo numérico y posteriormente a el.
- En el punto anterior da opción a un menú de ayuda que muestra todos los parámetros geométricos y las variables articulares de las siguientes estructuras robóticas de seis grados de libertad: antropoidal, cuadrilátero deformable, S.C.A.R.A., polar, cilíndrica, cartesiana, tipo Puma y tipo Standford. Esto se hace atendiendo a la posibilidad de que el usario trabaje sobre alguno de los tipos de estructuras mencionados.
- Permite la solución cinemática directa para cualquier manipulador de

hasta seis grados de libertad, dando la opción de calcular y mostrar:

1) La matriz numérica de transformación homogénea del elemento terminal
2) La matriz numérica de transformación del elemento i-ésimo de la estructura.
3) Los vectores Normal, Orientación, Aproximación y Posición del extremo final del manipulador.
4) Los mismos vectores anteriormente mencionados para cualquier elemento del manipulador.

- En cuanto a la generación de las expresiones simbólicas permite conocer a través de un menú de acceso las expresiones simbó licas de todos los términos de cada una de las matrices de transformación homogéneas del manipulador, dando la posibilidad de impresión de las mismas. Lo mencionado es válido tanto para manipuladores de estructura conocida como para nuevos diseños realizados por el usuario.

  Es de hacer notar también que el programa que desarrolla este cálculo simbólico opera en base a la generación autoregresiva de las expresiones simbólicas del manipulador.

- Permite además la impresión de resultados pudiéndose conocer cualquiera de las matrices de transformación homogénea del manipulador, así como los parámetros y variables articulares que se corresponden con tales resultados.

- En lo concerniente a la graficación brinda las siguientes posibilidades:

  1) ROBCAD inspecciona los parámetros geométricos ingresados en la opción y si estos se corresponden con alguna de las es-

tructuras disponibles en el sistema dibuja el perfil real de la misma en alta resolución (vista lateral de la estructura). Caso contrario dibuja un perfil genérico de una estructura industrial. En todos los casos se advierte al usuario si los datos se ajustan o no a alguno de los manipuladores.

Posteriormente dibuja los vectores de la mano del estremo del manipulador y explica su significado. Las estructuras arriba referidas son las siguientes: rotacional, S.C.A.R.A., cuadrilátero deformable, polar, cartesiana, cilíndrica y tipo Puma.

2) Sea cual fuere la estructura, permite obtener gráficos lineales de las tres vistas principales de los planos coordenados, es decir de los terminan el posicionamiento real de la estructura de acuerdo a la longitud de los parámetros geométricos ingresados. También se accede a un submenú que permite llamar una ayuda aclaratoria, quitarla e invertir el video de la vista cuando se desee.

3) Existe asimismo la opción de graficar las tres vistas en sendas ventanas en pantalla, con las bondades mencionadas en el punto anterior dando acceso a un submenu que permite:

   a) Ampliar el tamaño de cualquiera de ellas
   b) Invertir video de cualquiera de las vistas
   c) Restituir la pantalla original
   d) Imprimir la pantalla

4) El menú de graficación posibilita también obtener un croquis tridimensional de la posición del manipulador en el espacio, graficándose este de acuerdo a los datos que hayan

sido ingresados.

Este croquis incorpora un submenú de opciones que permite:

a) Llamar a un menu aclaratorio

b) Quitarlo y restituirlo

c) Invertir video de pantalla

d) Ingresar al modo aproximación o modo Zoom y salir de él.

Con referencia a este último punto el modo Zoom fue diseñado para permitir al diseñador inspeccionar un área del croquis tridimensional mediante un cursor que permite ser posicionado por toda la pantalla, seleccionando el punto deseado del dibujo para ser ampliado. Una vez seleccionado un punto del dibujo se accede a un nuevo submenú que permite optar entre seis niveles de aproximación diferentes, opción según la cual se grafica en una ventana en pantalla la región del espacio que rodea al punto elegido con el cursor, dibujándose la misma en una escala menor, logrando de este modo la aproximación deseada. Se puede salir de un nivel de aproximación e ingresar a otro diferente, abandonar un punto de inspección y seleciónar otro, así como invertir el video de la pantalla y el video de la aproximación por separado.

Estos constituyen los aspectos más destacados de las posibilidades ofrecidas al usuario en el marco de la solución al problema cinemático directo.

4.4 Descripción particular de la solución al Problema Cinemático Inverso.

4.4-1 Consideraciones teóricas

Los métodos utilizados comunmente en la solución de este proble-

ma presentan el inconveniente de no dar una indicación clara de como seleccionar una solución apropiada de las diversas soluciones posibles para una configuración de brazo en particular. Por esta razón se encaró un enfoque geométrico basado en el sistema de coordenadas de los elementos y la configuración del manipulador, para obtener una solución en forma cerrada. Con este método se logró solucionar el problema cinemático inverso para las estructuras admitidas por ROBCAD.

4.4-2 Posibilidades ofrecidas

La solución al problema cinemático inverso permite al usuario atender los aspectos relativos a la misma para siete estructuras de manipuladores industriales, considerando en la solución inversa de los mismos las distintas alternativas surgidas de la geometría propia del manipulador. Las estructuras para las cuales ROBCAD ofrece solución al problema cinemático inverso son las siguientes:

1) Estructura rotacional
2) Estructura cuadrilátero deformable
3) Estructura S.C.A.R.A.
4) Estructura tipo Puma
5) Estructura polar
6) Estructura cilíndrica
7) Estructura cartesiana

Las características más notables de la solución ofrecida al problema cinemático inverso son las siguientes:

- Una vez elegida una de las estructuras mencionadas posibilita el ingreso de los parámetros geométricos de la misma, dando la opción de confirmar y/o modificar estos cuantas veces lo desee el usuario.
- Incorpora en este punto un menú de ayuda que grafica en alta reso-

lución la vista lateral de una estructura real del tipo elegido, en el que se indica por un lado cuales parámetros del robot son fijos y cual es su valor, y por otro, cuales son pedidos al usuario, de manera que este tenga una idea acabada del dimensionamiento geométrico del robot.

- Asimismo si las coordenadas del punto espacial ingresado corresponden a un punto ubicado fuera de la región de acceso del manipulador, se advierte entónces esta situación al usuario, posibilitando que reingrese o modifique los valores citados.
- Permite la solución de la estructura elegida dando la opción de que esta posea de tres a seis grados de libertad, es decir calcula la solución de la misma considerándola sin mano o con mano de uno a tres grados de libertad, pudiendo solucionarse una misma estructura con distintas manos.
- Incorpora también otro menú que muestra una mano de tres grados de libertad indicando los valores de los parámetros geométricos que permanecen fijos y los que son solicitados al usuario, en un gráfico de alta resolución.
- Calcula la solución inversa contemplando las distintas variantes geométricas de las estructuras Polar, Cilíndrica y Cartesiana, dando la opción al operador de elegir la solución de la estructura con el brazo a la izquierda o a la derecha de la posición del hombro, o en su defecto una estructura de tipo planar. La misma alternativa se ofrece para la estructura de Tipo Puma, con las opciones de brazo a la izquierda y derecha de la posición del hombro.
- Advierte al usuario la existencia de más de una solución cinemática inversa, permitiendo calcular la misma con el codo arriba o abajo

de la posición de la muñeca del manipulador.

- Soluciona las singularidades y degeneraciones cinemáticas que pudieran presentarse, indicando al usuario la existencia de cualquiera de estas situaciones.
- Permite obtener las ecuaciones simbólicas que solucionan en forma numérica el problema cinemático inverso para las estructuras arriba mencionadas, dando la posibilidad de su impresión.
- Incluye la posibilidad de obtener la solución cinemática directa del manipulador inmediatamente obtenida la solución cinemática inversa, pudiendo de esta manera el operador verificar la exactitud de los resultados inversos logrados por el sistema.

  Esta solución directa tiene las mismas características que las descriptas en la solución al problema cinemático directo.
- Posibilita la impresión de los resultados obtenidos para una estructura determinada, imprimiendo también las coordenadas y parámetros geométricos introducidos. Se otorga además la opción de imprimir los resultados correspondientes a la solución directa, como verificación de exactitud, logrando disponer así de la documentación del estudio realizado.
- Es válido todo lo mencionado en la descripción de la solución a la cinemática directa en lo referente a la graficación para la solución cinemática inversa, pues incorpora los mismos procedimientos gráficos.

Estos aspectos constituyen las principales características desarrolladas en la solución al problema cinemático inverso implementada en ROBCAD.

A continuación se muestran dos pantallas que ejemplifican las alternativas mencionadas antes. La figura 1 muestra el

croquis tridimensional de una estructura tipo antropoidal de tres grados de libertad, habiéndose ingresado al modo aproximación. Puede observarse el cursor posicionado en un punto cercano al codo de la estructura, y el "zoom" de esta región utilizando el segundo de los seis niveles de acercamiento posibles.

La figura 2 en cambio, muestra una pantalla de la solución al problema cinemático inverso. En este caso se trata de una estructura del tipo cuadrilátero deformable en la que se ha solicitado

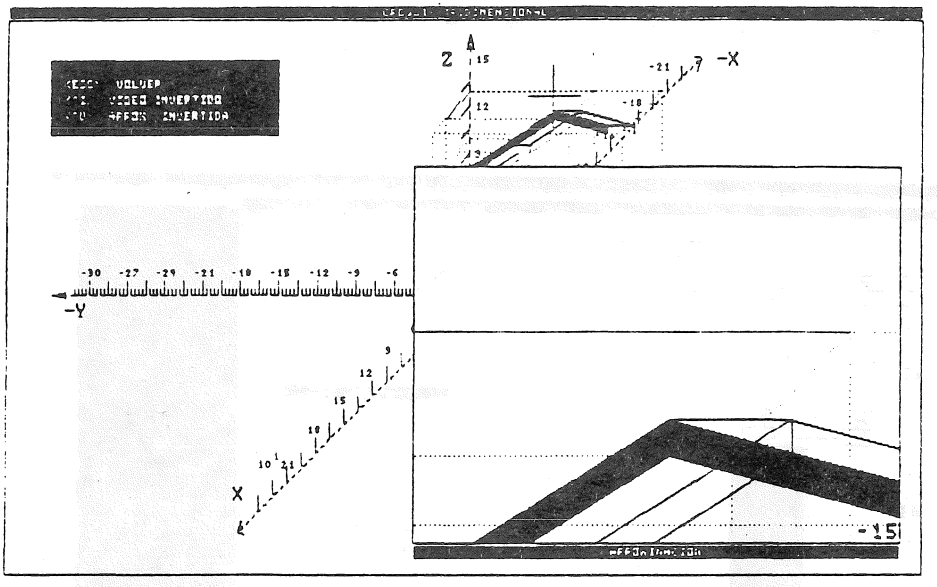


Figura 1: Pantalla con croquis tridimensional de una estructura del tipo antropoidal de tres grados de libertad. Puede observarse el acercamiento realizado.

el menú de ayuda referente al ingreso de los parámetros geométricos de la estructura.

5 Conclusiones finales

Se ha desarrollado un sistema de estudio y diseño cinemático se manipuladores de hasta seis grados de libertad, tratándolo de dotar de la mayor cantidad de posibilidades y recursos para el diseñador.

ROBCAD ha sido elaborado para ser ejecutado en cualquier ordenador personal del tipo IBM-PC o compatibles con el mismo, debido a la extendida difusión de este tipo de computadores personales. Corre bajo sistema operativo DOS (Disk Operating System), versión 2.00 o superiores a esta, y necesita que el computador posea targeta gráfica del tipo IBM, mónocromática Hércules o

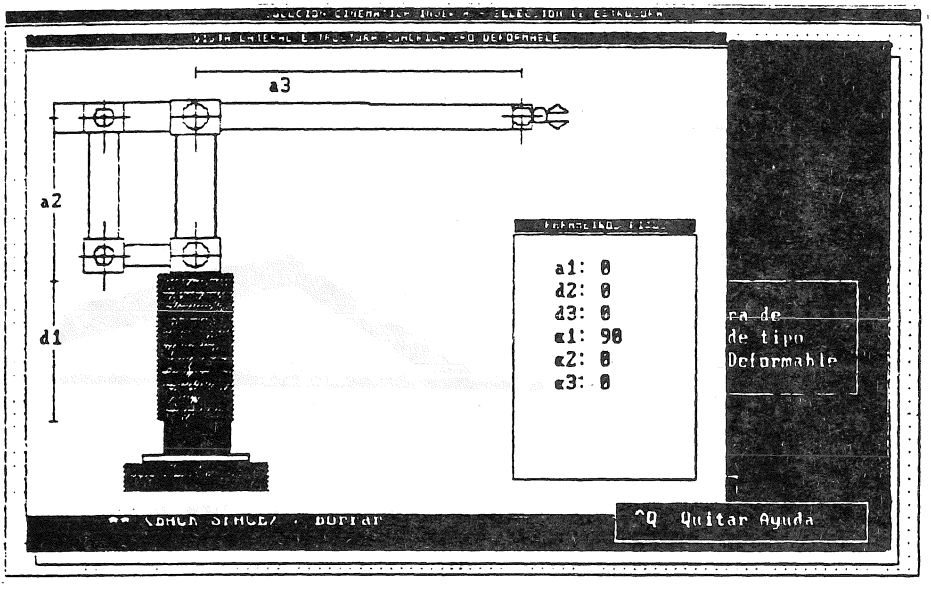


Figura 2: Pantalla de ayuda a la estructura cuadrilátero deformable en la solución al problema cinemático inverso.

Zhenith, o bien cualquier otra que emule a una de estas.

La solución al problema cinemático directo es una potente herramienta de cálculo para cualquier manipulador de hasta seis grados de libertad, mientras que la solución al problema cinemático inverso se ha restringido a la consideración de las siete estructuras industriales más usuales, teniendo en cuenta también sus variantes geométricas. Esta limitación esta dada por la misma naturaleza de la solución al problema cinemático inverso, puesto que la misma es abierta. Sin embargo, gracias a la modularidad de la que se ha provisto al sistema es posible el agregado de bloques que contengan la solución de nuevas estructuras, aumentando así la potencia informática de ROBCAD. Las alternativas de graficación implementadas complementan el diseño y estudio de un manipulador, permitiendo que el paquete cumpla con los propósitos trazados al comienzo de su formulación.

El sistema ha sido probado para una variedad extendida de casos y posibilidades, y en todos ellos se ha verificado la exactitud de los resultados, logrando ser así una confiable herramienta de estudio y cálculo cinemático de manipuladores robóticos.

Por su naturaleza la implementación efectuada contribuye amplia y originalmente al diseño y estudio de manipuladores robóticos, habiendo logrado este sistema un gran interés.

Referencias

[1] Introduction to Robotics. Mechanics and Control. John J. Craig 1986.

[2] Control of Manipulators Robots. Theory and Aplications.

Vukobratovic M, Stokic O., Springer Verlar 1982.

[3] Robot Manipulators. Mathematics, Programing and Control.
Paul R. 1981.

[4] Robot Arm Kinematics, Dynamics and Control.
Lee. C. IEEE Computer. Dic 1982.

[5] A Kinematic Notacion For Lower-Pair Mechanims based on Matrices.
Denavit Hartenberg. ASME
Journal of Applied Mechanics. Jun 1955.

[6] Advanced Mechanisms. Stilles J.

[7] Industrial Robots: Computer, Interfacing and Control. Wesley E. Snyder.
Prentice Hall 1985.

[8] Robótica: Control, Visión, Detección e Inteligencia. Fu, Gonzalez, Lee.
Mac Graw Hill. 1987.

[9] Análisis Cinemático de Mecanismos. Shigley J.

[10] Curso de Robótica. Angulo, Aviles. Paraninfo 1984.

BENCHREL - BENCHMARK PARA A AVALIAÇÃO DE DESEMPENHO DE SGBD'S RELACIONAIS

Maria Antonieta Rossatto (*)

Programa de Engenharia de Sistemas - COPPE
Universidade Federal do Rio de Janeiro
Caixa Postal 68.511
21.945 - Rio de Janeiro - RJ, Brasil

ABSTRACT

This work describes some aspects of benchmarks. Static and dynamic aspects of performance evaluation and fundamental features of benchmarks are presented. A methodology for development of benchmarks is described and a comparative study of the DBMS's is developed to evaluate their functionality. Finally, some benchmarks are implemented to evaluate the performance of the DBMS's. The results are examined and the criticals features of the DBMS's are identified.

(*) Eletrobrás - Centrais Elétricas Brasileiras S/A

I. INTRODUÇÃO

A área de banco de dados tem se desenvolvido, nas últimas décadas, de forma marcante. Novos SGBD's têm sido lançados no mercado, tornando grande a quantidade de produtos disponíveis e forçando possíveis usuários a escolher entre os que se diferenciam no tipo, objetivos, características, capacidade, desempenho, funcionalidade e custo. Entretanto, dificilmente, encontra-se um SGBD "ótimo" que atende todos os requisitos e resolve todos os problemas da instalação.

A análise de desempenho de um SGBD não deve estar associada ao objetivo de escolher o melhor, no sentido absoluto da palavra. Deve-se interpretá-la como um modo de escolher o melhor SGBD que atenda os requisitos da instalação e objetivos da empresa.

O programa de Engenharia de Sistemas da COPPE/UFRJ desenvolveu um SGBD relacional COPPEREL, que entrou em operação em 1983. Este é utilizado em pesquisas na universidade e serve de base para vários projetos. Entretanto, está incompleto e precisa ser otimizado para incorporar novas características. O desenvolvimento de um benchmark para avaliar o desempenho de SGBD's Relacionais permite analisar o COPPEREL e identificar as otimizações que devem ser feitas no mesmo.

II. METODOLOGIA PARA ANÁLISE DE DESEMPENHO DE SGBD'S

O desempenho de um SGBD depende de alguns fatores. Deve-se considerar parâmetros estáticos e dinâmicos. Parâmetros estáticos são, geralmente, usados para analisar funcionalidade. Parâmetros dinâmicos são, geralmente, usados para analisar tempo e implementados através do uso de benchmarks.

Existem SGBD's que possuem resultados de desempenho muito semelhantes e funcionalidades muito diferentes. Logo, é necessário analisar não somente os parâmetros dinâmicos, mas também os parâmetros estáticos para escolher um SGBD. A análise dos parâmetros estáticos deve preceder a análise dos dinâmicos. Uma vez conhecida a funcionalidade de um SGBD, torna-se mais fácil avaliar seu desempenho, pois são conhecidos recursos disponíveis, componentes integrantes e formas de utilização.

Definiu-se, então, uma metodologia composta de parâmetros estáticos e dinâmicos que será utilizada para avaliar o desempenho dos SGBD's e será apresentada a seguir.

II.1 PARÂMETROS ESTÁTICOS

Foram definidos os seguintes parâmetros estáticos: **Estrutura física e organização do espaço de armazenamento**, **Estrutura de dados**, **Dicionário de dados**, **Linguagem de definição e manipulação de dados**, **Interface com o usuário**, **Integração com outros SGBD's**, **Qualidade da documentação**, **Segurança e proteção dos dados**, **Deteção de falhas e reconstrução do banco de dados**, **Concorrência**, **Portabilidade e flexibilidade**, **Gerador de aplicações** e **Ambiente operacional**. Em alguns casos, torna-se necessário analisar a arquitetura do SGBD, e utilitários disponíveis. Assim, permite-se que instalações usem a potencialidade máxima de cada produto, adaptem-o as suas necessidades e otimizem suas tarefas. Informações adicionais podem ser obtidas em [Ross91].

II.2 PARÂMETROS DINÂMICOS

Benchmarks tem sido utilizados, com sucesso, na avaliação de desempenho de SGBD's. Os resultados de desempenho representam o desempenho dos produtos em relação a um determinado tipo de aplicação e não devem ser generalizados para todos os tipos de aplicações, pois um SGBD não se comporta igual em todas. Estas devem representar situações reais que ocorrem no ambiente em teste, simular o funcionamento do SGBD e gerar resultados de desempenho representativos e próximos da realidade da empresa.

Caso não seja possível codificar subpartes de uma aplicação na linguagem do SGBD, deve-se excluí-las dos benchmarks de alto nível que testarão cada produto. Deste modo, evita-se que os resultados gerados fiquem distorcidos e insignificantes quando comparados entre si. Deve-se incluir, somente, as transações implementáveis em todos os SGBD's num benchmark de alto nível.

A execução dos benchmarks nos SGBD's gera resultados de desempenho. Estes variam com a aplicação, recursos e ambiente operacional. Logo, devem representar o mais fielmente possível o desempenho do SGBD, pois, geralmente, servem de base para os profissionais de banco de dados escolher seus produtos.

Existem dois tipos de benchmarks para SGBD's: baixo nível e alto nível. Os de alto nível analisam SGBD's através da execução de uma aplicação completa e coleta dos resultados. Os de baixo nível analisam SGBD's através da execução de partes isoladas de uma aplicação e coleta dos resultados. Em SGBD's

relacionais, deve-se dividir uma aplicação em transações onde cada uma é composta de uma ou mais operações relacionais. Para cada transação, executa-se os benchmark de baixo nível, obtém-se resultados de desempenho e compara-se os SGBD's.

O benchmark deve ser simples, fácil de utilizar e requerer um esforço mínimo do usuário para sua execução. Simplicidade não implica em perda de poder como muitas pessoas pensam. É errado pensar que para analisar um SGBD complexo, deve-se utilizar um benchmark complexo. Existem benchmarks simples que analisam sistemas complexos de forma segura e eficiente.

III. METODOLOGIA PARA DESENVOLVIMENTO DE BENCHMARKS

A construção de benchmarks é uma tarefa difícil e complexa, que involve diversos fatores. A metodologia apresentada em [Ston85] serviu de base para a construção da metodologia que será apresentada. Esta metodologia será composta de três fases:

a. Projeto

Estabelece-se configuração e condições ambientais nos SGBD's em teste, aplicação e dados a serem utilizados, e parâmetros de comparação. Estes parâmetros contabilizam os desempenhos obtidos durante a execução do benchmark e medem tempo.

Após a definição dos parâmetros de comparação, estabelece-se uma forma real de medir tempo para computar os parâmetros, que permita a interpretação rápida e sem ambiguidade dos resultados de desempenho. Geralmente, utiliza-se o relógio da máquina hospedeira. A leitura deve ocorrer imediatamente antes e depois da execução de cada evento a ser analisado, e os resultados anotados para evitar perda de informações.

b. Execução

Executa-se o benchmark e obtém-se os resultados de desempenho. Inicialmente, deve-se executar o benchmark em modalidade monousuária, mesmo que o ambiente operacional em uso seja multiusuário. O processo deve ser repetido várias vezes, de forma a confirmar e garantir os resultados de desempenho.

Finalmente, caso o ambiente operacional seja multiusuário, executa-se o benchmark em modalidade multiusuária e obtêm-se o desempenho dos SGBD's em condições reais. Repete-se o processo diversas vezes, para confirmar os resultados. Assim, reduz-se a influência de fatores externos como compartilhamento dos dados do banco de dados, compartilhamento da CPU e dispositivos de E/S, multiprogramação, concorrência, paralelismo e deadlock.

c. Análise

Analisa e compara os resultados de desempenho de cada SGBD. Em ambientes multiusuário, deve-se considerar a influência de fatores externos nos resultados de desempenho, pois estes podem diferenciar o comportamento dos SGBD's nos dois ambientes.

Finalmente, escolhe-se o "melhor" SGBD para a instalação, com base nos parâmetros estáticos e dinâmicos. Durante a análise, o usuário deve ter clareza dos objetivos do benchmark e dos parâmetros de comparação. Assim, gera-se medidas de tempo que retratam interesses e necessidades do usuário, e servem para comparar características e desempenhos reais de SGBD's.

III.1 PARÂMETROS DE COMPARAÇÃO

Foram definidos alguns parâmetros que serão utilizados pelo benchrel para avaliar o desempenho dos SGBD's. A medida dos tempos depende do critério estabelecido na fase de projeto.

a. Tempo de resposta

É o intervalo de tempo entre a submissão da aplicação e a apresentação dos resultados, isto é, o intervalo de tempo entre a chamada do SGBD e apresentação dos resultados das consultas.

b. Tempo de Execução

É o intervalo de tempo entre o início e o fim do processamento. É o somatório do tempo de CPU com o de E/S. Não é considerado o tempo de espera por recursos em ambientes multiusuário.

c. Tempo de CPU

É o tempo de processamento.

d. Tempo de E/S

É o tempo de acesso aos dados situados no banco de dados.

e. Tempo de operação

É o intervalo de tempo entre o início e o fim do processamento de uma operação. Deve-se construir vários benchmarks de baixo nível que executem cada operação em separado, garantindo-se que cada operação não sofre influência de outras operações.

f. Tempo de improdutividade ("overhead")

É o tempo gasto pelo sistema para executar tarefas que não contribuem de forma líquida para o funcionamento do SGBD. Em [Turb87], a improdutividade é calculada executando-se uma "consulta vazia", isto é, sem comandos de CPU e E/S.

g. Produtividade ("Throughput")

É o volume de processamento realizado pelo SGBD num intervalo de tempo. Geralmente, a produtividade é medida em função do número de operações ou transações realizadas por segundo.

IV. ESCOLHA DOS SGBD's

Os benchmarks propostos serão utilizados para testar alguns SGBD's existentes no mercado de microcomputadores. Foram escolhidos 7 SGBD's relacionais: Copperel, dBase III/Plus, Dialog Plus/c, Oracle, Paradox, Rbase V, e Zim. A 'escolha resultou de pesquisas sobre os SGBD's. Foram mantidos contatos com representantes nacionais que se mostraram interessados na utilização destes em [Ross91].

Os resultados de desempenho gerados pelo benchmark precisam ser analisados. Para tanto, é necessário conhecer com precisão a funcionalidade de cada SGBD. Estudos dos produtos, permitiram avaliar suas características e capacidades, obedecendo os parâmetros estáticos fixados na metodologia. A seguir será apresentado um quadro comparativo da funcionalidade dos SGBD's.

QUADRO COMPARATIVO DA FUNCIONALIDADE DOS SGBD's

	COPPEREL	DBASEIII/PLUS	DIALOG PLUS/C	ORACLE
a.	Indexação por "Hashing"	Indexação por Árvore B+	Indexação por Árvore B+	Indexação por Árvore B+ e Concentração
b.	Caracter Numérico Inteiro Natural Data	Caracter Numérico Lógico Memo Data	Caracter Numérico Lógico Memo Data Inteiro	Caracter Numérico Data Binário
c.	Esquema	Catálogo	Catálogo	DD ativo
d.	Lopere]	dBase	Dialog	SQL*PLUS
e.	Interativa sem menus	Interativa por menus	Interativa por menus	Interativa por menus
f.	nenhuma	nenhuma	c/ DBASE	total c/SGBDs que usam SQL,
g.	português	inglês	português	inglês
h.	senhas níveis acesso	senhas níveis acesso	senhas níveis acesso	senhas níveis acesso
i.	Journal Image AI e BI files	sem mecanismo restaurador	sem mecanismo restaurador	Journal Image AI e BI files
j.	Bloqueio	Bloqueio	Bloqueio	Bloqueio
k.	Mono e Multiusuário	Mono e Multiusuário	Mono e Multiusuário	Mono e Multiusuário
l.	PC, UNIX, IBM, UNISYS,	PC	PC, UNIX, Cobra	Todos os ambientes

Figura IV.1

QUADRO COMPARATIVO DA FUNCIONALIDADE DOS SGBD's (Cont.)

	PARADOX	RBASE SYSTEM V	ZIM
a.	Identificação por chaves	Indexação por Árvore B+	Indexação por Árvore B+
b.	Caracter Numérico Inteiro Data Monetário	Caracter Numérico Inteiro Data Hora Monetário	Caracter Numérico Inteiro Data
c.	DD ativo	DD ativo	DD não ativo
d.	PAL	Rbase/SQL	ADL
e.	Interativa sem menus	Interativa por menus	Interativa por menus
f.	c/DBASE	c/DBASE c/SGBDs que usam SQL	c/DBASE c/SGBDs que usam SQL
g.	inglês	inglês	inglês
h.	senhas níveis acesso	senhas níveis acesso	senhas níveis acesso
i.	restauração parcial	sem mecanismo restaurador	Journal Image AI e BI files
j.	Bloqueio	Bloqueio	Bloqueio
k.	Mono e Multiusuário	Mono e Multiusuário	Mono e Multiusuário
l.	PC, IBM, VAX	PC	PC, IBM, VAX, COBRA, UNIX,

Figura IV.2

V. ESPECIFICAÇÃO, PROJETO E EXECUÇÃO DOS BENCHMARKS

A especificação, projeto e desenvolvimento dos benchmarks tomou como base a metodologia proposta. Estes utilizaram aplicações reais em seus testes para melhor simular o funcionamento dos SGBD's na instalação em uso e, gerar resultados de desempenho mais representativos e próximos da realidade. Entretanto, em ambientes com vários tipos de aplicações, a construção de um benchmark para cada tipo, é uma tarefa custosa. A solução foi a construção de uma aplicação única, simples e genérica que englobou os requisitos e objetivos daquelas da instalação.

De forma a permitir o desenvolvimento de um benchmark representativo de ambientes reais de banco de dados, foram analisadas as aplicações desenvolvidas na Eletrobrás Centrais Elétricas Brasileiras S. A., e observadas as operações mais frequentes. Apesar dos SGBD's utilizados não serem exatamente os mesmos que serão testados nesta tese, as transações envolvidas em aplicações de banco de dados independem do SGBD em uso e empregam combinações de operações de forma a atender seus objetivos. Entretanto, a amostra obtida é restrita e dependente das aplicações analisadas, e contém um conjunto reduzido de transações, sendo representativa somente para os ambientes considerados, não devendo ser generalizada para todos os ambientes de banco de dados.

Para construir a amostra de transações foram estabelecidas classes de transações típicas, de forma a distribuir os valores encontrados por classe e apurar a frequência de cada uma dentro da amostra. Para tanto, definiu-se uma transação típica Ti como uma combinação de operações sobre relações. Cada Ti usa relações como operandos e produz uma relação como resultado. A seguir, serão definidas as transações Ti.

T1 = Seleção
T2 = Projeção
T3 = União
T4 = Intersecção
T5 = Diferença
T6 = Divisão
T7 = Produto Cartesiano
T8 = Junção
T9 = T1 + T8 = Seleção + Junção
T10 = T5 + T3 = Subtração + União
T11 = T1 + T5 + T3 = Seleção + Subtração + União
T12 = T1 + T2 = Seleção + Projeção
T13 = T1 + T5 = Seleção + Subtração
T14 = T1 + T2 + T8 = Seleção + Projeção + Junção
T15 = T1 + T3 = Seleção + União
T16 = T14 + T10 = Seleção + Projeção + Junção + Subtração + União
T17 = T14 + T5 = Seleção + Projeção + Junção + Subtração
T18 = T14 + T3 = Seleção + Projeção + Junção + União

A escolha destas transações deve-se ao fato de serem estas as mais utilizadas pelas aplicações da instalação analisada. Uma análise detalhada das aplicações da Eletrobrás permitiu gerar uma distribuição de frequência. A seguir, serão apresentados quadros com o número de transações encontradas por aplicação, num total de 3.165 transações e 17 aplicações (P1 ... P17).

Os resultados obtidos permitiram concluir que as transações T1, T3 e T14 foram as mais freqüentes, com percentagens de 30.4%, 20.6% e 11.8%, respectivamente. A seguir, será apresentado um gráfico comparativo acumulativo (figura V.3) da freqüência das transações no ambiente da Eletrobrás. No eixo das abscissas, serão representadas as classes de transações, que são 18, e no eixo das ordenadas serão marcadas as freqüências acumuladas, cuja maior ocorrência é 963 e corresponde a 30.4% do total.

DISTRIBUIÇÃO DE FREQUÊNCIA DAS TRANSAÇÕES TÍPICAS POR APLICAÇÃO

T/A	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
T1	23	103	2	50	80	54	91	9	18	20
T2	3	2	3	2	2	13	8	3	6	8
T3	10	44	4	83	20	45	141	33	81	21
T4	6	10	2	12	8	9	16	1	3	11
T5	2	9	4	0	7	1	3	3	0	2
T6	0	0	0	0	0	0	0	0	0	0
T7	0	0	0	0	0	0	0	0	0	0
T8	61	4	0	7	0	10	0	3	0	1
T9	2	2	0	5	2	2	0	3	0	1
T10	0	0	0	0	0	0	0	0	0	0
T11	21	8	0	0	20	3	11	37	0	4
T12	3	1	1	1	0	0	5	1	0	7
T13	1	3	1	0	20	3	1	5	0	1
T14	0	35	2	0	7	0	11	30	1	62
T15	3	11	0	13	0	1	16	0	1	2
T16	11	1	0	7	0	2	9	0	2	13
T17	7	1	2	0	3	0	11	0	0	1
T18	5	1	0	5	0	1	7	1	1	13
TOTAL	158	235	21	185	169	144	330	129	113	167

Figura V.1

DISTRIBUIÇÃO DE FREQUÊNCIA DAS TRANSAÇÕES TÍPICAS POR APLICAÇÃO (Cont.)

T/A	P11	P12	P13	P14	P15	P16	P17	%	TOTAL
T1	15	63	121	31	189	46	40	30,4	963
T2	5	15	12	5	6	25	9	4,0	127
T3	73	40	32	2	2	3	17	20,6	651
T4	5	12	20	1	13	12	6	4,6	147
T5	3	6	2	0	1	4	2	1,6	49
T6	0	0	0	0	0	0	0	0,0	0
T7	0	0	0	0	0	0	0	0,0	0
T8	0	10	4	5	82	13	0	6,3	200
T9	0	1	9	0	4	0	1	1,0	32
T10	0	0	0	0	0	0	0	0,0	0
T11	0	21	41	0	3	10	17	6,2	196
T12	3	11	6	1	3	13	2	1,8	58
T13	0	6	15	0	21	3	1	2,6	91
T14	0	1	20	0	184	17	2	11,6	372
T15	0	63	4	4	12	3	7	4,4	140
T16	0	3	1	3	0	5	1	1,8	58
T17	0	6	10	0	1	0	1	1,4	43
T18	0	6	0	3	0	4	1	1,5	48
TOTAL	104	264	297	55	521	158	115	100	3165

Figura V.2

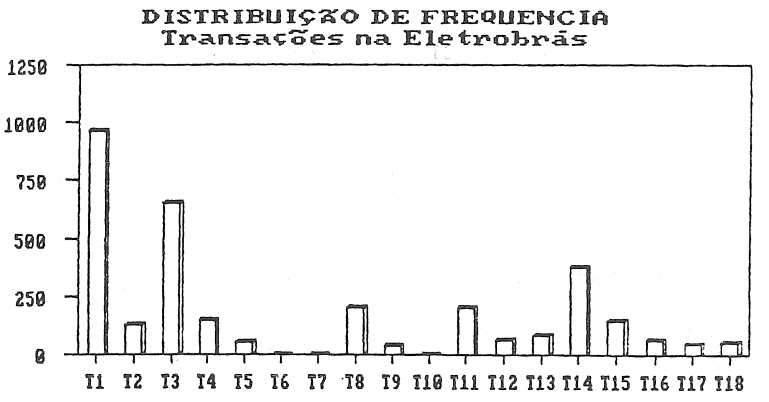


Figura V.3

Tornou-se necessário, então, definir uma aplicação para fazer parte do benchmark, que utilizasse todas as transações típicas definidas anteriormente, com distribuição de frequência igual a apresentada na amostra. A solução foi utilizar uma aplicação da Eletrobrás, pois esta estaria projetada e especificada no mesmo ambiente operacional e seria representativa da amostra obtida. Entretanto, suas aplicações são muito extensas e um benchmark requer uma aplicação pequena, simples e concisa que consiga realizar todos os testes de desempenho necessários à avaliação. Logo, decidiu-se trabalhar com uma subparte de uma aplicação que atendesse os objetivos e requisitos obrigatórios.

A aplicação escolhida foi a AEI (Aquisição de Equipamentos de Informática). Esta controla a aquisição de equipamentos de informática. A seguir, será apresentada a subparte do diagrama entidade-relacionamento (E-R) estritamente necessária à demonstração dos conceitos deste trabalho. Foram suprimidos os atributos e chaves para não sobrecarregá-lo.

DIAGRAMA ENTIDADE-RELACIONAMENTO DA APLICAÇÃO AEI

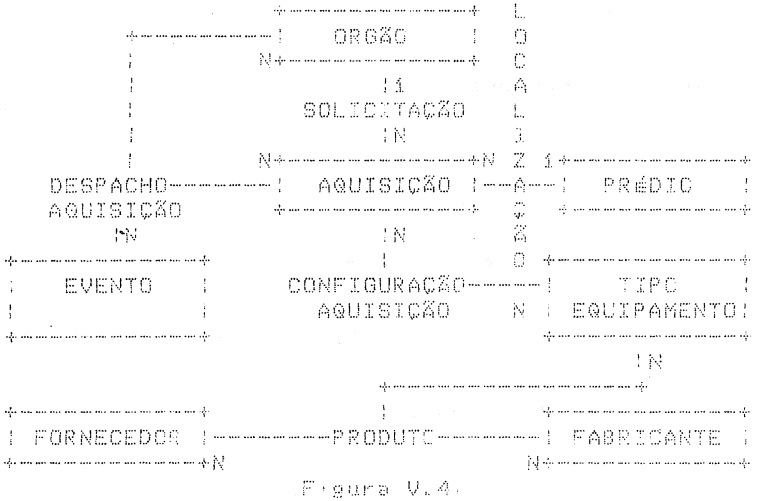


Figura V.4

As transações típicas Ti foram implementadas nas linguagens dos SGBD's em teste, utilizando as entidades da aplicação AEI. Cada Ti associada aos parâmetros de comparação fixados na metodologia, originou um benchmark de baixo nível, que foi executado nos SGBD's usando o módulo NI ("Norton Integrator") do "Norton Utilities" para medir o tempo imediatamente antes e depois da execução de cada evento a ser analisado.

Estes benchmarks foram executados juntamente com os SGBD's, em um microcomputador do tipo IBM-PC/AT operando na modalidade monousuária. As limitações no escopo do ambiente considerado visou isolar os efeitos causados pelas configurações de hardware, características do sistema operacional, softwares utilizados e fatores externos.

O processo de execução dos benchmarks foi repetido 5, 10, 100 e 1000 vezes, para confirmar os resultados e gerar parâmetros mais precisos e seguros. A execução de cada benchmark 1000 vezes verificou a estabilização dos parâmetros a medida que o número de execuções aumentou e, obteve um tempo de operação mais próximo do real. A cada mudança de SGBD e benchmark, foi zerado o buffer de memória para evitar a propagação de sujeiras e a influência de outros fatores no teste corrente.

A entrada dos dados da aplicação e dos comandos dos benchmarks foi feita em massa através da leitura de blocos de registros via arquivo externo, visando reduzir o tempo de improdutividade que seria gasto para selecionar opções em menus e, digitar e fornecer informações para os SGBD's via tela.

A cada execução de um benchmark em um SGBD, foram analisados os parâmetros dinâmicos fixados na metodologia. Entretanto, neste ambiente operacional deve-se fazer algumas considerações:

a. O **tempo_de_execução** coincidiu com o tempo de resposta, pois este ambiente é monousuário e não existe espera por recursos.

b. Durante a análise do tempo de execução, não foi possível distinguir o tempo de CPU do tempo de E/S, por não existir um procedimento para interceptar os acessos a disco e medir a E/S.

c. O **tempo_de_improdutividade** retratou o intervalo de tempo entre a chamada do SGBD para inicialização e carga do banco de dados, e o término da execução dos procedimentos necessários a construção do ambiente para teste dos benchmarks. Para tanto, foi executada uma consulta vazia, de forma a melhor retratar tempos improdutivos envolvidos na execução de uma transação.

d. Fixou-se o **tempo_de_operação**(Toper) como o intervalo de tempo necessário para processar uma transação Ti. Logo, Toper coincidiu com o tempo de execução de cada benchmark de baixo nível (descontado o tempo de improdutividade), pois cada benchmark representa uma transação típica Ti.

e. Fixou-se a **produtividade** como o número de transações Ti realizadas por segundo. Logo, esta foi calculada pela fórmula 1/Toper (1 dividido pelo tempo de operação).

f. Para a maioria das transações Ti, aumentando-se o número de execuções, o tempo total de execução e operação tendeu a aumentar uniformemente de t e o tempo médio tendeu a um valor constante e próximo do T anterior. Os tempos que serão apresentados nos quadros, são tempos médios.

g. Seja **produtividade_esperada**, (PEsp) o produto da frequência pela produtividade de cada transação Ti em cada SGBD. Assim, definiu-se o parâmetro **produtividade_média_esperada** que é obtido pelo somatório do PEsp de cada transação típica Ti.

A execução dos benchmarks em cada SGBD gerou resultados de desempenho. A partir dos tempos de operação de cada transação Ti em cada SGBD gerou-se um quadro comparativo, mostrado a seguir, para analisar o desempenho de cada produto em Ti.

Os resultados obtidos permitiram concluir que:

i. Ti a é a mais rápida na maioria dos SGBD's e T6 pode ser implementada, somente, no COPPEREL e ORACLE.

ii. O COPPEREL e o ORACLE são os únicos SGBD's com recursos que permitem implementar todas as operações relacionais. Não houve necessidade de simular as operações através da construção de procedimentos em linguagem de terceira geração. Os comandos disponíveis foram desenvolvidos com propósitos relacionais e, assim, foram suficientes para a implementação de todas as Ti.

QUADRO COMPARATIVO DO TEMPO DE OPERAÇÃO DE CADA TRANSAÇÃO TÍPICA EM CADA SGBD (em segundos)

Ti/SGBD	COPPEREL	DBASE	DIALOG	ORACLE	PARADOX	RBASE	ZIM
Tia	0,486	0,143	0,235	0,563	0,400	0,180	0,613
Tib	0,807	0,603	0,468	3,200	1,200	1,357	0,957
T2	0,878	0,532	0,487	2,400	–	1,592	–
T3	2,317	20,920	15,171	17,343	1,321	4,563	1,378
T4	0,745	4,026	5,672	1,995	–	3,387	1,378
T5	2,325	–	–	13,678	0,817	2,137	1,378
T6	2,985	–	–	0,380	–	–	–
T7	1,792	7,286	3,881	347,580	–	33,527	–
T8	1,593	1,593	1,600	7,727	–	2,138	2,006
T9	6,973	75,844	53,018	4,235	–	5,674	3,201
T10	0,801	0,368	0,266	0,451	–	1,158	1,433
T11	0,801	1,111	0,811	0,745	–	0,957	1,001
T12	1,371	0,610	0,600	0,917	–	0,563	–
T13	0,403	0,979	0,641	0,611	0,526	0,956	1,285
T14	6,981	75,830	53,018	1,454	–	5,676	1,823
T15	2,527	3,000	1,400	10,213	1,961	8,237	1,433
T16	7,912	133,611	71,999	0,577	–	7,273	2,375
T17	3,417	–	–	4,222	–	5,853	–
T18	4,678	3,251	2,999	14,743	–	5,861	–
Total	49,792	329,707	212,266	433,034	6,225	91,089	20,261

Figura V.5

iii. O PARADOX é o SGBD cuja linguagem possui menor poder relacional. Consequentemente, conseguiu-se implementar, somente, seis (6) transações em PAL.

iv. O COPPEREL é um SGBD que possui desempenho semelhante em todas as transações. Seu desempenho não ficou prejudicado em nenhuma delas. Por exemplo, na base de dados em teste, o tempo de operação foi inferior a 8 (oito) segundos.

v. O ORACLE teve seu tempo de operação prejudicado pela incorporação do tempo de improdutividade relativo a impressão dos resultados das consultas. Não desenvolveu-se procedimentos para interceptar as impressões e medir este tempo.

A partir da aplicação AEI, da distribuição de frequência das aplicações da Eletrobrás e dos benchmarks de baixo nível, foi definido e implementado um benchmark de alto nível T que englobou as transações Ti em um único programa. O benchmark T foi, também, executado várias vezes. Assim, analisou-se os resultados de desempenho de cada SGBD.

A partir destes resultados, construiu-se um quadro comparativo do tempo de operação médio de T em cada SGBD. A seguir, será apresentado este quadro com os valores de Toper em segundos. Deve-se considerar que algumas transações não puderam ser implementadas em T devido a deficiências de alguns produtos, e geraram tempos de operação distorcidos e insignificantes.

QUADRO COMPARATIVO DO TOPER DE T EM CADA SGBD (em segundos)

	COPPEREL	DBASE	DIALOG	ORACLE	PARADOX	RBASE	ZIM
T	113,102	1073,735	831,773	393,983	57,547	206,636	81,878

Figura V.6

Os resultados obtidos permitiram construir um gráfico comparativo do desempenho de T em cada SGBD. A seguir, este será apresentado. No eixo das abscissas, serão representados os SGBD's, que são 7(sete), e no eixo das ordenadas será marcado o tempo de operação de cada SGBD com relação a T.

GRÁFICO DO TEMPO DE OPERAÇÃO DE T EM CADA SGBD (em segundos)

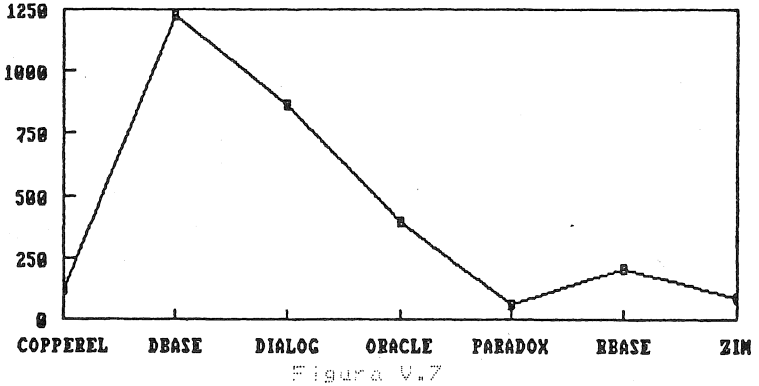


Figura V.7

A construção e implementação do benchmark de alto nível T ficou limitada pelas restrições impostas por alguns produtos. A falta de recursos de linguagem impediram a incorporação de algumas Ti em T. A seguir, será apresentado um quadro do número de transações implementadas e não implementadas em cada SGBD.

QUADRO COMPARATIVO DA IMPLEMENTAÇÃO DAS TRANSAÇÕES POR SGBD

SGBD	NO. TRANSAÇÕES IMPLEMENTADAS	No. TRANSAÇÕES NÃO IMPLEMENTADAS	TRANSAÇÕES NÃO IMPLEMENTADAS
COPPEREL	19	00	----
DBASE	16	03	T5, T6, T17
DIALOG	16	03	T5, T6, T17
ORACLE	19	00	----
PARADOX	06	13	T2,T4,T6,T7,T8, T9,T10,T11,T12, T14,T16,T17,T18
RBASE	18	1	T6
ZIM	13	6	T2, T6, T7, T12 T17, T18

Figura V.8

Pelo fato do PARADOX permitir a implementação de apenas 6 transações, a execução do benchmark de alto nível neste SGBD apresentou o menor tempo de operação. Este fato comprova que, em alguns casos, um benchmark de alto nível pode gerar resultados de desempenho distorcidos. Neste caso, o produto com menor capacidade relacional apresentou-se mais eficiente. Consequentemente, estes resultados e o gráfico não são válidos para analisar o desempenho dos SGBD's.

Deste modo, conclui-se que a análise de desempenho destes SGBD's no ambiente em questão deve considerar, principalmente, os resultados gerados pelos benchmarks de baixo nível e ignorar aqueles gerados pelo de alto nível T. Pode-se comparar, analisar e validar os resultados de desempenho obtidos de T, somente, para o COPPEREL E ORACLE, pois estes produtos são os únicos capazes de implementar e executar todas as transações Ti, gerando resultados comparáveis e significativos.

Adicionalmente, decidiu-se construir um benchmark de alto nível Tmin contendo o subconjunto máximo de Ti implementáveis em todos os SGBD's. Este é formado por 5 das 6 transações implementadas no PARADOX. Não foram utilizados recursos externos, mas somente recursos e linguagens disponíveis nos produtos, para implementar os benchmarks. Assim, o benchmark Tmin incorporou apenas as transações T1a, T1b, T3, T13 e T15.

O benchmark Tmin ficou limitado e não retratou com exatidão a aplicação AEI e a distribuição de frequência das aplicações na Eletrobrás. Entretanto, gerou resultados de desempenho comparáveis e significativos. A seguir, será apresentado um quadro comparativo do toper médio de Tmin em cada produto.

QUADRO COMPARATIVO DO TOPER DE Tmin EM CADA SGBD (em segundos)

	COPPEREL	DBASE	DIALOG	ORACLE	PARADOX	RBASE	ZIM
T	53,243	326,576	216,239	321,531	53,427	112,999	53,666

Figura V.9

Estes resultados mostraram que o toper obtido por Tmin foi bem próximo ao Toper obtido por T no PARADOX. Assim, comprovou-se que os números de T estavam distorcidos e insignificantes. A seguir, será apresentado um gráfico comparativo do desempenho de Tmin nos SGBD's. No eixo das abscissas serão representados os SGBD's e no eixo das ordenadas os toper médios.

Deste modo, pode-se concluir que o COPPEREL é o SGBD com maior poder relacional e melhor desempenho nos parâmetros dinâmicos, apesar de possuir deficiências nos parâmetros estáticos.

GRÁFICO DO TEMPO DE OPERAÇÃO DE Tmin EM CADA SGBD (em segundos)

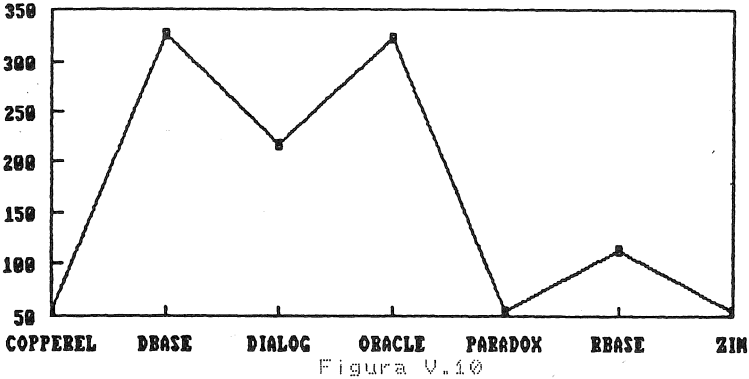


Figura V.10

VI. CONCLUSÃO

Não existe benchmark perfeito, que possa ser usado para avaliar o desempenho de todos os SGBD's em todas as instalações e não há receita para um benchmark bem sucedido. Não conseguiu-se desenvolver uma fórmula capaz de funcionar, perfeitamente, em todos os SGBD's, instalações e ambientes operacionais. Contudo, existem benchmarks suficientemente rigorosos para garantir algum grau de sucesso na maioria dos produtos analisados. Adicionalmente, deve-se mencionar que benchmarks permanecem como a única maneira prática de estimar o desempenho de SGBD's.

A implementação e execução dos benchmarks permitiu identificar características e deficiências de cada SGBD. Não há um que seja o "melhor" em todos os parâmetros estáticos e dinâmicos.

Finalmente, deve-se concluir que um SGBD gerado no curso de informática da U.F.R.J., sem fins comerciais, possui desempenho muito bom e funcionalidade relativamente boa. Não encontrou-se um produto que fosse muito superior ao COPPEREL. Portanto, após

algumas otimizações, este estaria em condições de competir com os melhores SGBD's relacionais no mercado de informática.

VII. FUTUROS TRABALHOS

Poderiam ser desenvolvidas novas estatísticas baseadas em amostras maiores de transações reais em uso em vários empresas. A nova amostra deveria conter um conjunto vasto de transações, seria representativa de ambientes reais de banco de dados e permitiria o desenvolvimento de benchmarks de alto nível mais completos. Estes poderiam ser utilizados para analisar o desempenho de grande parte dos SGBD's em vários ambientes operacionais com diversos tipos de aplicações. Para tanto, deveriam ser contactadas diversas empresas, de forma a analisar suas instalações e aplicações, observar as transações, montar a estatística e identificar transações típicas mais frequentes.

Os testes de desempenho poderiam ser extendidos para ambientes multiusuário. O benchmark deveria ser adaptado para considerar e analisar a influência dos fatores externos. Provavelmente, novos parâmetros dinâmicos deveriam ser definidos. Os testes precisariam ser regerados e os resultados reanalisados.

A metodologia ("benchmark") desenvolvida para avaliação do desempenho de SGBD's foi testada em 7 produtos. Contudo, esta poderia ser usada para avaliar outros SGBD's. Logo, deveria-se implementá-la utilizando a linguagem do produto em teste.

VIII. REFERÊNCIAS BIBLIOGRÁFICAS

[Bitt83] Bitton, Dina & DeWitt, David J. & Turbyfill, Carolyn, "Benchmarking Database Systems: A Systematic Approach", Proceedings of the 1983 Very Large Database Conference, Florence, Italy, 1983.

[Bora84] Boral, Haran & DeWitt, David J., "A Methodology for Database System Performance Evaluation", Proceedings of the 1984 ACM Sigmod International Conference on the Management of Data, Boston, June 18 - 21, 1984.

[DeWi85] DeWitt, David J., "Benchmarking Database Systems: Past efforts and Future Directions", Database Engineering, V8, N1, March 1985.

[Ston85] Stonebraker, Michael, "Tips on Benchmarking Data Base Systems", Database Engineering, V8, N1, March 1985.

[Ross91] Rossatto, Maria Antonieta Rossatto, "BENCHREL - Benchmark para a Avaliação de Desempenho de SGBD's Relacionais - e Propostas de Melhorias para o COPPEREL", Tese de Mestrado em Engenharia de Sistemas e Computação, COPPE/U.F.R.J., Rio de Janeiro, 1991.

[Turb87] Turbyfill, Carolyn, "Comparative Benchmarking of Relational Database Systems", Ph. D. Thesis, Number 87 - 871, Department of Computer Science, Cornell University, Ithaca, New York, September 1987.

UM SISTEMA DE INTERFACE COOPERATIVO PARA MODELAGEM EM BANCO DE DADOS

Luís Eduardo Saraiva Câmara Décio Fonseca
Departamento de Informática - UFPE
Caixa Postal 7851, 50739 Recife - PE

RESUMO

Este artigo apresenta um sistema de interfaces cooperativo e inteligente desenvolvido para facilitar o processo de modelagem de aplicações de Banco de Dados utilizando o modelo de dados E/D [1]. A ênfase maior está no tratamento de erros que possibilita o uso do modelo por um grande número de usuários de diferentes níveis de conhecimento

1 - INTRODUÇÃO

A necessidade de interfaces inteligentes para ambientes de concepção de Banco de Dados englobando ferramentas diversas e utilizados por usuários de diferentes níveis de conhecimento para permitir o uso simplificado e adaptável dos ambientes, se torna a cada dia mais vital [8, 12, 14, 15]. Pesquisadores de áreas diversas como Inteligência Artificial, Banco de Dados, Engenharia de Software, Psicologia Cognitiva e Lingüística Computacional, concordam que cada vez mais se torna necessário a existência de interfaces cooperativas que se ajustem ao conhecimento das pessoas que utilizam o ambiente [5, 9, 16]. Existe, também, uma corrente de pensamento que defende o desenvolvimento de tais interfaces através de uma cooperação interdisciplinar.

Muitos dos sistemas desenvolvidos no passado e que são utilizados atualmente, apresentam o problema da adaptabilidade das interfaces ao nível de compreensão do usuário em relação às funções do sistema. Nesses sistemas, o tratamento de erros é muito precário. Não existe a preocupação de interpretar o que foi solicitado por um comando não totalmente correto. A indicação do erro, geralmente, se restringe a mensagens e códigos de retorno cujo significado, muitas vezes, não é compreensível à maioria dos usuários.

Um método bastante comum para obter informações e/ou realizar operações, é a utilização de uma Interface de Linguagem

de Comandos. Tal interface é obtida através de uma linguagem formal que permite ao usuário expressar de forma não ambígua suas requisições ao sistema. O grande problema está no fato que este tipo de interface implica em uma grande familiaridade com a sintaxe da linguagem de comandos e com as funções que o sistema oferece. Para usuários leigos ou com pouca experiência, as interfaces tradicionais de Linguagem de comandos demonstram ser bastantes ineficientes.

Com a intenção de resolver estes problemas, diversos tipos de interfaces para usuários de menor grau de experiência têm sido desenvolvidos. Interfaces Gráficas, Interfaces baseadas em menus, Interfaces de linguagem Semi-natural eliminam ou suavizam as dificuldades existentes na comunicação entre os usuários leigos e o sistema. Estas interfaces requerem pouca sofisticação computacional e todo comando é solicitado pelo usuário, através de opções ou do preenchimento de respostas pré-definidas. A necessidade de um conhecimento prévio das funções do sistema é eliminada, uma vez que elas são apresentadas sempre que necessário. Os erros são diminuídos, visto que tais interfaces são projetadas de maneira a guiar a interação e sua indicação é feita de forma mais amigável.

Entretanto, à medida que usuários leigos adquirem experiência sobre o sistema através de sua utilização, torna-se claro que comandos relativamente simples exigem um longo processo de escolhas e respostas para serem realizados. Para estes usuários com uma nova visão sobre o sistema bem como para aqueles que já são especialistas, tais interfaces são consideradas monótonas e requerem muito tempo para a execução de comandos.

Na tentativa de se obter uma interface que sirva de maneira satisfatória tanto para usuários leigos como para usuários especialistas, pesquisas têm sido direcionadas para a interface de Linguagem natural. Claramente uma linguagem de comandos é uma sublinguagem da linguagem natural, e desta forma interfaces de Linguagem Natural não apresentam inconvenientes para os especialistas do sistema. Para os usuários leigos, o fato de poder dialogar com o sistema da mesma forma em que se expressam no dia-a-dia, torna bem mais amigável sua comunicação com o

sistema. Embora Interface de Linguagem natural seja uma boa alternativa para resolver o problema existente quando um sistema possui uma variedade de tipos de usuário, é reconhecido que o processamento de linguagem natural, na sua forma geral (principalmente, quando expressa por usuários leigos) requer um grande investimento de pesquisa, além do fato da língua natural ser, ela própria, ambígua.

O nosso trabalho constituiu-se na definição e implementação de um Sistema de interfaces interativo e convergente para a concepção de aplicações de Banco de Dados utilizando o Modelo Dinâmico e Estático (Modelo E/D). Dentro deste processo convergente, está embutida uma metodologia para a concepção interativa, evolutiva e cooperativa de Banco de Dados. A grande ênfase, nesta direção, está no tratamento de erros. Acreditamos que o tratamento inteligente e cooperativo de erros permite, principalmente, que :

- O ambiente seja utilizado por usuários de diferentes níveis de conhecimento;
- Usuários com experiência em outros modelos conceituais de dados, trabalhem satisfatoriamente com o modelo E/D;
- O manuseio do modelo E/D constitua-se, ele próprio, em um processo gradativo de aprendizado

O trabalho está dividido em 4 seções, das quais esta é a primeira. Na próxima seção são descritas as principais características do modelo Estático e Dinâmico. Na seção 3 é apresentado o Sistema de Interface com suas características cooperativas e convergentes. Finalmente, na última seção, são descritas algumas conclusões sobre o trabalho e aspectos necessários para a sua continuação.

2 - O MODELO ESTÁTICO DINÂMICO (E/D)

O modelo Estático e Dinâmico (E/D) é uma extensão do modelo Entidade / Relacionamento [7] com características de orientação a objetos. O modelo E/D foi desenvolvido na UFPE e integra o projeto POESIS cuja meta é a construção de um protótipo de ambiente de desenvolvimento de Banco de Dados multimídia [6]. O modelo E/D tem como objetivo principal tratar e especificar, de forma integrada, as propriedades estáticas, relativas aos estados (aspecto estrutural), as propriedades dinâmicas relativas às

transações de estado (aspecto comportamental) e o conjunto de estados válidos (restrições de integridade) dos sistemas de informação [1, 13].

O modelo E/D trata todos os seus conceitos como objetos. Cada objeto identificado no processo de modelagem é especificado através de uma Linguagem de Definição de Objetos (LDO). Os conceitos podem ser agrupados em tres grupos: Representação estrutural do modelo (Entidade, Relacionamento, Atributo, Domínio e respectivas restrições de integridade); Propriedades Dinâmicas (Ação e Regra de Comportamento) e propriedades de Controle (Transação e Meta-regra).

ENTIDADE é o objeto básico do modelo E/D. Suas propriedades estruturais são representadas pelos atributos e as comportamentais por ações e regras de comportamento. Uma classe entidade tanto pode herdar propriedades de outras classes entidades (herança) quanto ser definida a partir de outras classes (Generalização Especialização). O modelo comporta herança múltipla e herança seletiva [2]. Uma classe entidade deve possuir, no mínimo, uma propriedade que a identifique (atributo determinante). Sua especificação compreende: Identificação, Descrição, Atributos, Restrições de Integridade, Ações, Regras Comportamentais, Generalização e Especificação.

ATRIBUTO representa as propriedades relativas às classes entidades e às associações entre elas (relacionamento). É um objeto que está associado a um conjunto de valores (instâncias). A especificação de um objeto atributo compreende: Identificação, Descrição, Tipo (simples ou composto), Multivaloração, Domínio, Sinônimo, Edição, Restrição e Determinante.

DOMÍNIO representa um contexto ou uma estrutura sob a qual um atributo está definido. Pode ser do tipo mais básico (inteiro, caracter, boleano) como de estruturas mais complexas (voz, imagem, registro). A especificação de cada objeto domínio compreende: Identificação, Descrição e Tipo.

RELACIONAMENTO é uma estrutura que indica as associações entre instâncias de uma classe entidade com outra, ou ainda, com ela mesma (auto-relacionamento). É um objeto dependente, ou seja, sua existência só faz sentido se for para representar associações entre classes entidades. O relacionamento é um objeto que, embora

possa ter propriedades próprias, não pode herdar propriedades de classes entidades ou de outros relacionamentos. Sua especificação compreende: Identificação, Descrição, Participantes, Atributos próprios, Restrições de Integridade, Ações, Regras de Comportamento, Forma de associação (Total ou Parcial).

RESTRICÕES DE INTEGRIDADE evitam que os objetos fiquem em um estado incorreto ou inconsistente. O modelo E/D trata as restricões de integridade como um conjunto de objetos. Através deste conjunto, asserções são definidas sobre os objetos entidade, atributo e relacionamento. As restrições de integridade são classificadas em: Pré-definidas (Dependência Funcional, Unicidade, Não Nulidade, Dependência de Inclusão Excludente), Verificação e Cardinalidade.

AÇÃO permite descrever os fatos que modificam os estados dos objetos entidade e relacionamento através de um conjunto de operações básicas (inserção, Modificação, Recuperação, etc). Sua especificação integrada às propriedades estruturais aumenta a capacidade de representação do modelo. O objeto Ação é composto de: Identificação, Descrição, Entidade e/ou Relacionamento e a Ação.

REGRA DE COMPORTAMENTO representa as propriedades comportamentais dos objetos entidade e relacionamento. Este objeto permite exprimir uma ação condicional composta de uma regra que envolve uma condição para que o fato/ação ocorra e a ação propriamente dita. Sua especificação compreende: Identificação, descrição, Entidade e/ou Relacionamento e a regra.

TRANSAÇÃO caracteriza um procedimento atômico envolvendo duas ou mais ações e/ou regras de comportamento que modificam os objetos entidade e relacionamento. Sua especificação compreende: Identificação, Descrição, Entidade e/ou Relacionamento e Conjunto de Procedimentos.

META-REGRA representa o mecanismo de controle na aplicação das regras de comportamento. O objeto meta-regra permite definir prioridades de execução entre as regras, controlar a execução de regras mutuamente excludentes e controlar a dependência de uma ou mais regras em relação a outras. Sua especificação compreende: Identificação, Descrição e Conjunto de Controle.

3 - O SISTEMA DE INTERFACES

O Sistema de Interface cooperativo, cuja arquitetura é apresentada na figura 1, foi desenvolvido utilizando a linguagem PROLOG e compreende, basicamente, cinco módulos: Meta-base de Conhecimento, Modelo do Usuário, Analisador Sintático, Aquisição de Conhecimento, Explicativo. Estes módulos, atuando em conjunto, são capazes de fornecer ao usuário do modelo E/D as seguintes facilidades:

- Adaptação da interface ao usuário;
- Definição e evolução do perfil do usuário;
- Explicações sempre que requisitado;
- Tratamento amigável de erros;
- Aquisição de conhecimento.

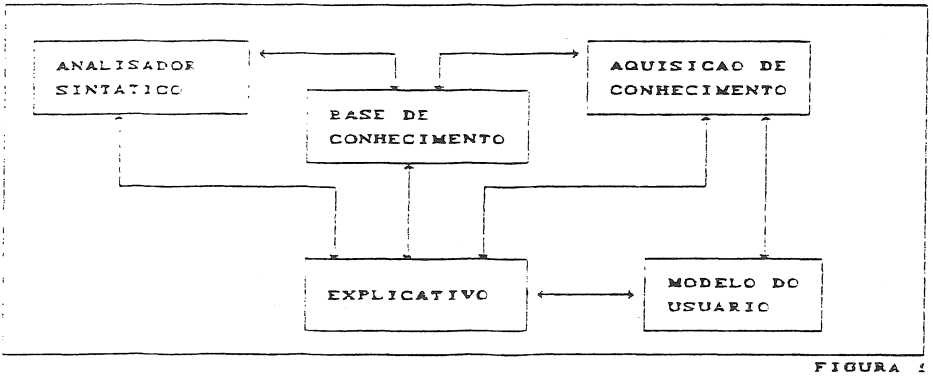


FIGURA 1

Antes de descrever com maiores detalhes as funções do Sistema de Interface, será apresentada a estrutura da Meta-base de conhecimento, uma vez que ela serve como fonte de alimentação para a maioria das funções.

3.1 - META-BASE DE CONHECIMENTO

A meta-base de conhecimento é uma base de regras que visa representar o conhecimento necessário ao Sistema de Interfaces cooperativo para a modelagem de Banco de Dados (modelo E/D, usuários e aplicações). Os objetos da meta-base são modelados a partir do próprio modelo E/D.

A meta-base de conhecimento é composta pelos objetos: entidade, relacionamento, atributo Estas informações torna

possível o funcionamento do tratamento de erros, definição do perfil, explicações e serve como alimentação para o analisador sintático. Na figura 2 são apresentadas as definições dos principais objetos que serão necessários no contexto do trabalho.

O objeto entidade COMANDO é composto de quatro atributos. O atributo *COMANDO* pode ser dividido em dois subatributos. Isto ocorre em virtude de alguns comandos E/D necessitarem de duas palavras chaves para serem identificados. O atributo *MAX-PARA* estabelece o número máximo de parâmetros do comando. O atributo *QTD-PRE* indica os códigos dos comandos que são pré-condições e que devem ser satisfeitas para a execução do comando. É um atributo multivalorado.

O objeto-relacionamento PARAMETRO é responsável pela formação completa do comando. A partir de seus dados, o Analisador Sintático é capaz de, através de casamento de padrões, assegurar a corretude de um comando enviado. Sua concepção é baseada na Forma Normal de Backus - BNF em que cada expressão é quebrada em subexpressões para uma melhor representação.

```
OBJETO-ENTIDADE   : COMANDO
DESCRICAO         :/* Entidade representando os comando do
                   modelo E/D
ATRIBUTOS         : COMANDO, CODIGO, MAX-PARA, QTD-PRE
FIM-OBJETO-ENTIDADE

OBJETO-RELACIONAMENTO : PARAMETRO
DESCRICAO             :/* Autorelacionamento, descrevendo os
                       parametros de cada comando E/D
PARTICIPANTES         : CLASSE-COMANDO(CODIGO)
ATRIBUTOS             : SEQUENC, OPCIONAL, PAL-RES, STRING,
                        OCOR-MUL, CARC-ESP
RESTRICAO             : OBJETO-CARDINALIDADE(UM-PARA-MUITOS)
RELACIONAMENTO-TOTAL  : NAO
FIM-OBJETO-RELACIONAMENTO

OBJETO-RELACIONAMENTO : SINONIMO
DESCRICAO             :/* Relacionamento, associando cada
                       comando E/D a um sinonimo
PARTICIPANTES         : CLASSE-COMANDO(CODIGO)
ATRIBUTOS             : SINONIMO
RESTRICAO             : OBJETO-CARDINALIDADE(UM-PARA-UM)
RELACIONAMENTO-TOTAL  : SIM(CLASSE COMANDO)
FIM-OBJETO-RELACIONAMENTO
```

FIGURA 2

A conjunção *CODIGO* e *SEQUENC* identifica qual a posição do parâmetro em relação ao comando. O atributo *OPCIONAL* indica se o

parâmetro é ou não opcional. O atributo *PAL-RES* é preenchido se o parâmetro for uma palavra reservada da sintaxe do comando, ou seja, não é uma informação fornecida pelo usuário. De forma inversa, o atributo *STRING* informa qual a categoria da informação dada. Entre as categorias válidas estão: atributo, entidade, relacionamento, índice, variável ou código do subcomando, caso este parâmetro corresponda a uma expressão. Expressões são tratadas da mesma forma que comandos. A indicação se o parâmetro é de ocorrência múltipla ou não, é feita pelo objeto-atributo *OCOR-MUL*. Dependendo da BNF do comando, existem alguns parâmetros que são caracteres especiais. O conjunto de caracteres especiais é representado pelo objeto-atributo *CARC-ESP*. A figura 3 traz a representação gráfica do comando DEFINE DOMINIO e sua respectiva representação na meta-base.

O objeto-relacionamento SINÔMINO, associa um comando E/D a um comando estrangeiro. Comandos estrangeiros são comandos associados a um modelo de dados qualquer, e que possui um correspondente no modelo E/D. O Sistema de Interface tem cadastrado para cada comando E/D, os comandos mais conhecidos em outros modelos de dados no objeto-relacionamento SINÔNIMO público. Cada usuário, entretanto, pode requerer ao sistema seus próprios sinônimos. Seu cadastramento é concretizado através do diálogo com cada usuário e são armazenados no seu objeto relacionamento SINÔNIMO privado. Na aquisição de novos sinônimos, uma checagem simples de verificação torna-se necessária para manter consistente e coerente a base de conhecimento.

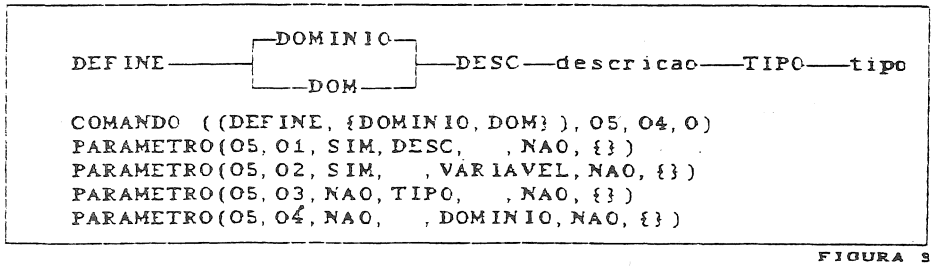


FIGURA 3

3.2 - ADAPTAÇÃO DA INTERFACE, DEFINIÇÃO E EVOLUÇÃO DO PERFIL DO USUÁRIO

A função de adaptação da interface é realizada usando as

informações existentes no módulo MODELO DO USUÁRIO. Iniciada uma seção, o sistema de interface recupera o perfil do usuário que possui todos os dados referentes ao seu grau de conhecimento do modelo E/D. Estes dados ajudam na escolha da forma de comunicação mais adequada para que os objetivos do usuário sejam alcançados.

Para novos usuários do sistema, uma pseudo-entrevista é estabelecida. Neste processo, as informações necessárias para a composição do perfil podem ser obtidas de uma forma amigável e descontraída. A pseudo-entrevista é baseada na abordagem de estereotipos [11] que consiste, fundamentalmente, em um conjunto de caracteristicas que coexistem nos usuários do ambiente e que podem, a partir de um número pequeno de observações (no nosso caso através de perguntas), ser atribuídas ao entrevistado. No momento atual, o Sistema de Interface dispõe de duas formas de comunicação: Interface baseada em menus e a Interface de linguagem de comandos. Associada a cada tipo estão, respectivamente, os usuários leigos e os não leigos. A possibilidade de uma classe tão geral como a dos não leigos é resultado do bom tratamento de erros que possibilita usuários de perfis diferentes fazerem uso da mesma interface, embora com grau de assistência e forma de cooperação, diferenciados.

A evolução do perfil do usuário é essencial para o nosso sistema de interfaces, visto que a aquisição de conhecimento é um processo contínuo. À medida que o usuário se torna familiar com as funções existentes no sistema e a ocorrência de erros atinja a um patamar desprezível, é necessário que novas formas mais avançadas de interação com o ambiente sejam oferecidas a ele.

O processo de aprendizado e evolução do conhecimento têm demonstrado ser uma tarefa bastante complexa e difícil e requerem, dentro do contexto deste trabalho, uma forte ligação entre Informática, Educação e Psicologia Cognitiva. Este trabalho está sendo iniciado juntamente com os Departamentos de Letras, Eduacação e Psicologia [3]

Em [4] é retratado um sistema assistente de projeto físico de Banco de Dados com tratamento diferenciado para duas classes de usuário: leigo e especialista. Neste trabalho, mesmo tratando com grupos tão distintos, a detecção da evolução do conhecimento continuou sendo bastante difícil. Foi constatado que a mudança de

um perfil para outro sempre requer a necessidade da definição de um perfil intermediário. O critério adotado consistiu em atribuir pesos a cada tipo de solicitação de explicação e tipo de erro cometido. Após um número pré-estabelecido de interações, o usuário, através de seus pontos, era ou não transferido para outro perfil. Apesar de sua importância, a abordagem da evolução do perfil, em razão das dificuldades encontradas, segue, neste trabalho a mesma linha de [4].

3.3 - SISTEMA DE APOIO AO USUÁRIO

A função que permite ao sistema fornecer explicações à medida em que for solicitado é executada pelo módulo EXPLICATIVO utilizando as informações a respeito do perfil do usuário e da base de conhecimento. O usuário tem a possibilidade de questionar o sistema a qualquer momento, independente do seu grau de conhecimento e da interface utilizada no momento. Neste contexto, a explicação consiste no resultado de uma solicitação de informações adicionais a respeito do estado atual ou do comando a ser executado.

Estando a função de explicação disponível a qualquer usuário, uma mesma requisição pode ser respondida de várias maneiras diferentes. O perfil do usuário determina o formato (sentenças em linguagem natural, seqüência de comandos na linguagem de comandos, gráficos, etc) e a complexidade (detalhista, generalizada, superficial) da explicação. A seguir, cada forma de explicação e sua forma de implementação são detalhadas.

3.3.1 - EXPLICAÇÃO DO ESTADO

Este tipo de explicação encontra-se disponível tanto na interface baseada em menus como na de linguagem de comandos. Sua ativação é feita através de uma tecla especial (F1). Considerando que a interface baseada em menu é destinada a usuários completamente leigos, e que o processo de escolhas pode implicar na passagem de diversos submenus, a importância da explicação compreende:

- Situar o usuário em seu trabalho, explicando porque foi necessário a passagem de outros estados para se atingir o estado atual;
- Caso o estado atual não corresponda a um estado final,

fornecer detalhes sobre o significado dos estados finais que podem ser atingidos do ponto atual.

Em relação à interface de linguagem de comandos, a explicação abrange apenas o último comando enviado aceito e inclui informações funcionais.

3.3.2 - EXPLICAÇÃO DO COMANDO

Explicação de comandos está diretamente associada a Interface de linguagem de comandos, e por conseguinte, a usuários especialistas. Por especialista, considera-se aqueles que possuem conhecimento do modelo E/D e aqueles que têm trabalhado com Sistemas de Banco de Dados, porém não, necessariamente, com o modelo E/D.

Para evitar que a segunda classe de especialistas seja classificada como usuário leigo e tenha que modelar suas aplicações através de menus, foi criado um mecanismo que permite a correspondência entre comandos E/D e comandos estrangeiros. Enviado um comando estrangeiro ao sistema, o módulo Explicativo retorna a sintaxe completa do comando E/D correspondente e sua explicação funcional.

Toda cadeia de caracteres precedida por uma interrogação é considerada uma requisição de explicação de comando. Solicitada a explicação, o Sistema pesquisa o comando na seguinte ordem: Objeto entidade COMANDO, objeto relacionamento SINÔNIMO público e objeto relacionamento SINÔNIMO privado. Caso a pesquisa não tenha sucesso, o módulo Explicativo é transferido para o módulo Aquisição de Conhecimento para a inclusão de um novo sinônimo.

Estabelecido qual comando E/D ou seu correspondente foi requerido, o Analisador Sintático gera a sintaxe completa do comando. Neste ponto do diálogo, o usuário é capaz de prosseguir sua modelagem. Na figura 4 é apresentado, parcialmente, o objeto-relacionamento SINÔNIMO público referente ao comando 01, DEFINE ENTIDADE.

```
SINONIMO(DEFINE ENTIDADE, {CREATE TABLE,  DEFINIR,  ENTIDADE,
                           CRIACAO, CRIAR TABELA, GERACAO})
```

FIGURA 4

3.4 - TRATAMENTO AMIGÁVEL DE ERROS

A função de tratamento de erros é realizada em conjunto pelos módulos meta-base de conhecimento e analisador sintático. O tratamento amigável de erros não é aplicável a todo tipo de interface. [10] afirma que certas formas de comunicação homem-máquina como interfaces gráficas ou baseadas em menus, são projetadas de forma a interagir com o usuário, isentas de erros. Não existe, nestes tipos de interação, a possibilidade de se requisitar um comando cujas pré-condições não tenham ainda sido satisfeitas. Em nosso trabalho, a análise de erros está restrita às interfaces de linguagem de comando.

O tratamento amigável de erros corresponde a não se rejeitar completamente o comando quando algum tipo de erro estiver associado a ele. De acordo com o tipo do erro, o sistema pode: corrigí-lo sem a interferência do usuário, requisitar informações adicionais e não um novo envio do comando reformulado ou alertar o usuário que alguns comandos devem ser executados anteriormente ao comando enviado.

Durante uma interação com o ambiente utilizando uma linguagem de comandos, diversos tipos de erros estão propensos a ocorrer. Os erros podem ser categorizados em: expressão, sintaxe, associação e preparação.

3.4.1 - ERROS DE EXPRESSÃO - Correspondem aqueles surgidos na digitação do comando. Em muitos sistemas, erros de expressão implicam na resubmissão do comando, mesmo que estes se encontrem no final de um comando de muitos parâmetros. O Sistema de Interface detecta estes tipos de erros de forma a agilizar a utilização do modelo E/D. Como existe uma ordem de precedência de cada parâmetro de um comando, sempre que ocorre um problema de reconhecimento, por exemplo a inexistência de um comando ou de um objeto-entidade, o analisador enquadra este não reconhecimento como erro de expressão. Como a maioria dos erros de digitação são de duplicação e troca de caracteres, duas soluções são tentadas para validar o parâmetro com erro de expressão: Eliminação de caracteres adjuntos repetidos e varredura exaustiva do parametro desconhecido, onde a cada processamento, um caracter é desconsiderado na pesquisa.

3.4.2 - ERROS DE SINTAXE - Ocorrem quando o usuário sabe que tipo de comando deve ser executado, porém uma parte ou toda a sintaxe do comando não é lembrada. O analisador sintático tenta estabelecer um casamento de padrões entre o comando enviado e a sintaxe armazenada. Havendo discordância, apenas parametros *STRING* obrigatórios omitidos são solicitados para completar o comando. Sempre que um parâmetro *PAL-RES* que delimita dois tipos *STRING*, e um destes string forem omitidos, é possível que o Sistema de Interface entre em conflito, pois diversas instanciações do comando são geradas. Neste caso, o usuário deve intervir e informar qual das interpretações corresponde à sua solicitação.

O tratamento de erros de sintaxe permite que a linguagem de comandos possa ser manuseada por usuários não leigos de diferentes perfis, pois a necessidade do conhecimento prévio da sintaxe do comando é eliminada. Aqueles com pouca familiaridade com o modelo E/I, necessitam apenas enviar o nome principal do comando, seja ele E/I ou estrangeiro. O sistema de interface pede os parâmetros necessários através de um diálogo amistoso. Quanto mais especialista for o usuário, mais completo são seus comandos e menor é a intervenção do sistema. Através de seus erros e omissões, o usuário avança no aprendizado da linguagem.

3.4.3 - ERROS DE ASSOCIAÇÃO - São erros de origem semântica e que correspondem a relacionamentos errôneos entre objeto-atributo e respectivos objeto-entidade ou objeto-relacionamento. Quando de uma discordância, o Sistema de Interface tenta estabelecer, usando sua meta-base, que outras associações entre atributo, entidade e relacionamento, são possíveis:

- Objetos entidade ou objetos relacionamento que tenham como propriedade o objeto atributo informado;
- Objetos atributo associados ao objeto entidade ou objeto relacionamento informado, possuidores do mesmo domínio semântico do objeto atributo informado.

3.4.4 - ERROS DE PREPARAÇÃO - Ocorrem quando o comando é considerado compreensível ao sistema, porém não é possível executá-lo, pois algumas pré-condições devem ser satisfeitas. O sistema informa que o comando não pode ainda ser processado e

indica que comandos necessitam ser enviados antes do seu processamento. Está sendo projetado, embora não implementado, um módulo de PLANOS/OBJETIVOS, que possibilitará ao sistema deduzir, através das solicitações do usuário, qual o seu objetivo, ou seja, que estado se deseja alcançar. Estabelecido o objetivo, é deduzido o plano escolhido para esta tarefa. Este módulo possibilitará que comandos, incompatíveis com a situação atual de interação, sejam detectados e haja um bom tratamento para esta classe de erros.

4 - CONCLUSÃO

O sistema de Interfaces foi desenvolvido como uma conseqüência natural do Sistema Assistente para Projeto Físico de Banco de Dados [4]. Seu objetivo inicial foi facilitar o processo de modelagem através do modelo E/D. No momento atual, o Sistema de Interface está sendo incorporado ao ambiente POESIS e embora ainda existam algumas restrições, o Sistema de Interfaces tem conseguido ser utilizado eficazmente por pessoas de diferentes níveis e ser um instrumento de auxílio e aprendizado.

O Sistema de Interface deverá ser estendido para abranger outros módulos do ambiente POESIS, como o Integrador de Visões e o Especificador Formal. Para tanto, a Meta-base de conhecimento necessitará de outros objetos Entidade e Relacionamento para representar as novas informações.

Acreditando que o tratamento de erros é o grande responsável pela possibilidade de diversos tipos de usuários usarem satisfatoriamente o ambiente, novas formas de cooperação, principalmente para erros de associação e preparação, devem ser investigadas. Um estudo mais aprofundado da evolução do conhecimento do usuário torna-se, também, necessário. Este estudo possibilitará a existência de outros perfis de usuário e formas de interface.

5 - REFERÊNCIAS BIBLIOGRÁFICAS

[1] M. Bandeira. Modelagem Estática/Dinâmica de Sistemas de Informações. Tese de mestrado, UFPE, 1989.

[2] J. Banerjee, W. Kim, K. Kim. Queries in Objects-Oriented Databases. Object-Oriented Database I, IEEE, 1988.

[3] F.A Barros. Aspectos Teóricos e Práticos no Reconhecimento

Automático de Fenômenos Linguísticos. Tese de Mestrado, UFPE, 1990.

[4] L.E.S. Câmara. Um Sistema Assistente para Projeto Físico de Banco de Dados. Relatório Técnico. Departamento de Informática, UFPE, 1990.

[5] J.M. Carroll, J.Mckendree. Interface design issues for advice giving expert systems. Communications of the ACM. 30(1), 1987.

[6] P. Chen. The Entity-Relationship Model: Toward an unified new date. ACM Transaction on Database System. 1(1), 1976.

[7] D. Fonseca. Relatório do Projeto APOIO. Relatório técnico. Departamento de Informática, UFPE, 1990.

[8] H.R. Hartson, Hix D. Human-Computer Interface Development : Concepts and Systems for its Management. ACM Computing Surveys. 21(1), 1989.

[9] R. Hirschheim, H.K. Klein. Four paradigms of Information Systems development. Communication of the ACM. 32(10), 1989.

[10] E. Kantorowitz, O. Surarsky. The adaptable user interface. Communications of the ACM. 32(11), 1989.

[11] A. Kobsa, W.Wahlster. User Model in Dialog Systems. Springer-Verlag. 1989.

[12] A. Motro. FLEX: A Tolerant and Cooperative User Interface to Databases. IEEE Transactions on Knowledge and Data Engineering. 2(2), 1990.

[13] R. Motz. Especificação Formal para o Modelo E/D. Tese de mestrado. UFPE. 1990.

[14] J.M. Pager, D.M. Lamberti, D.L. Gardner, S.R. Balzac. REASON An intelligent user assistant for interactive environments. IBM Systems Journal. 29(1), 1990.

[15] E. Schoen, R.G. Smith, B.G. Buclanam. Design of Knowledge-Based systems with a Knowledge-Based assistant. IEEE Transactions on Software Engineering. 14(12), 1988.

[16] R. Zwicker, N. Reinhard. Interfaces Inteligentes: Perspectivas para novas formas de aprendizado e uso de sistemas. Revista Brasileira de Computação. 5(3), 1990.

Acumulación y Desacumulación de Evidencia en Sistemas de Mantenimiento de la Verdad

Ramón García Martínez - Antonio Marsiglio

PROGRAMA DE INVESTIGACION Y DESARROLLO EN INTELIGENCIA ARTIFICIAL
UNIVERSIDAD NACIONAL DE LUJAN

Abstract : In this paper we present the results of statiscal experiments on Non-monotonic Truth Maintainance Systems (NTMS) . The results allowed us to stablish some observations over the behavior of the NTM .

Resumen : En este trabajo presentamos los resultados de un estudio estadístico sobre los Sistemas No-monotónicos de Mantenimiento de la Verdad (SNMV) a partir de los cuales hemos postulado un conjunto de observaciones sobre el comportamiento de los mismos.

1. Introducción :

Entre las aplicaciones de Inteligencia Artificial existen sistemas que acumulan evidencia o donde las premisas una vez establecidas no se modifican, es decir, sistemas monotónicos de mantenimiento de la verdad . La observación del mundo real induce a pensar que este tipo de modelos no siempre se adecua correctamente a la realidad . La cuestión es saber que ocurre en los sistemas de manipulación de conocimiento cuando premisas que en un determinado instante son ciertas en el siguiente son falsas o que en un instante acumulan evidencia a favor de su grado de verdad y en otro desacumulan evidencia en contra de su grado de verdad. Es evidente que todo el sistema de premisas deducido a partir de la premisa mutada puede llegar a carecer de sustento (por ejemplo : si despúes de una desacumulación la evidencia asociada a la premisa en cuestión queda por debajo de la cota mínima de verdad del sistema) por lo que no puede estar contenido en el conjunto de los conceptos conocidos , en este caso se está en presencia de un Sistema No-monotónico de Mantenimiento de la Verdad (SNMV). En el contexto de los sistemas de mantenimiento de la verdad [1] se encuentran los procesos cíclicos de acumulación de evidencia [2,3,4,5] o propagación cíclica de evidencia. En el formalismo de representación de conocimiento mediante reglas, un proceso cíclico de acumulación de evidencia esta definido por un encadenamiento de reglas donde el antecedente de la primera regla es el consecuente de la última. Los procesos cíclicos de acumulación de evidencia pueden ser estables o inestables [6,7], en los primeros los hechos que forman parte de la cadena deductiva estabilizan la evidencia asociada después de una cantidad finita de ciclos, en los segundos algunos hechos no logran estabilizar la evidencia asociada. Se define como nodo saturado al hecho para el cual nuevos ciclos no agregan nueva evidencia.

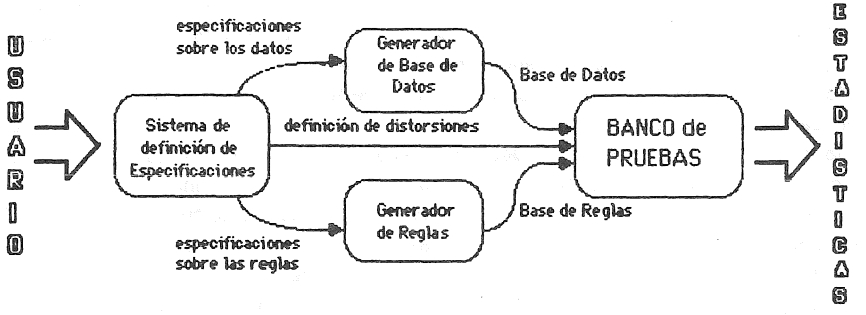
2. El Problema :

Uno de los sistemas de mantenimiento de la verdad de uso más frecuente en la Inteligencia Artificial es el asociado a la representacion del estado del conocimiento en los Sistemas Expertos .

La mayoría de los S.E. trabajan con un sistema monotónico de representación de la verdad o del valor de verdad de las premisas del universo de discurso del cuerpo de conocimiento involucrado [8]. Que ocurriría en estos sistemas si permitiésemos que el usuario acumulase evidencia en contra de una premisa que antes había sido establecida como verdadera o que al seguir distintas líneas de razonamiento exploradas por el motor de inferencia este llegase a una contradicción . De hecho por la forma en la cual los S.E. son construidos no puede ocurrir la primera circunstancia y por la forma en la cual son diseñados los entornos de desarrollo de S.E. no puede ocurrir la segunda, pero que no ocurran no necesariamente nos lleva a concluir que no se debe permitir que ocurran . Muchos dominios de conocimiento trabajan sobre lo que se conoce como el arquetipo de las líneas hipotéticas [9,10]. Las líneas hipotéticas parten del supuesto de que un determinada premisa es cierta , a partir de esta postulación se concluyen nuevas premisas si el proceso deductivo concluye con éxito se acumula evidencia sobre la premisa presuntamente cierta, si por el contrario se llega a una contradicción se desacumula evidencia sobre la premisa original y si la evidencia asociada a esta queda por debajo de la cota mínima de verdad del sistema, todas aquellas premisas que se dedujeron de esta no pueden considerarse a primera inspección como verdaderas .

3. El Banco de Pruebas :

El banco de pruebas diseñado para el testeo de los SNMT es parte de uno mayor sobre el cual se experimenta en Sistemas Expertos . Se determinó que el banco original podía ser extendido para el tratamiento del estudio estadístico de varios tipos de soporte de conocimiento , en particular el basado en reglas . los módulos que conforman el sistema general y sus interrelaciones pueden visualizarse en el siguiente esquema :



El sistema tiene cuatro módulos básicos :

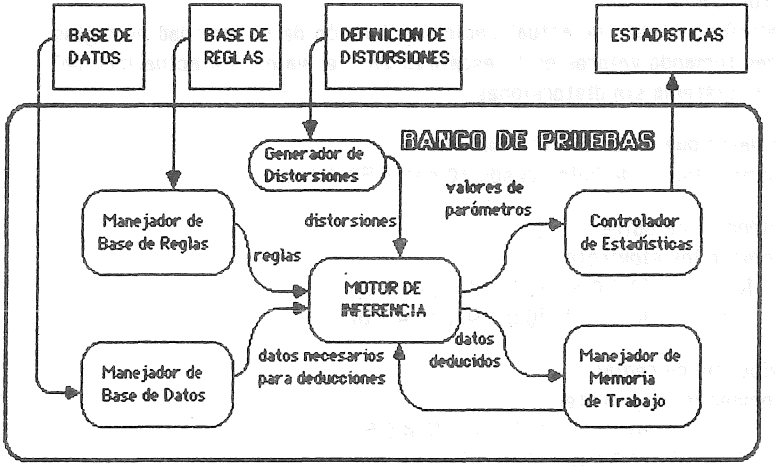
[a] Sistema de Definición de Especificaciones : que interactuando con el usuario define características del sistema de representación de conocimiento como tipo de reglas, soporte de evidencia, modelo de acumulación y propagación de evidencia , parámetros de la Base de Reglas, parámetros de la Base de Datos y tipos de distorsiones en el modelo de mantenimiento de la verdad .

[b] Generador de Base de Datos : genera en forma aleatoria una Base de Datos según especificaciones dadas por el Sistema de Definición de Especificaciones .

[c] Generador de Base de Reglas : genera en forma aleatoria una Base de Reglas según especificaciones dadas por el Sistema de Definición de Especificaciones .

[d] Banco de Pruebas : activa la base de reglas a partir de la información contenida en la base de datos y memoria de trabajo , llevando estadísticas de los parámetros indicados por el usuario .

Un detalle del Banco de Pruebas se ofrece en el siguiente diagrama :



El Banco de Pruebas tiene cinco módulos básicos :

[a] Manejador de Base de Reglas : lleva cuenta del estado de las reglas que en función de su disparabilidad pueden estar habilitadas o inhabilitadas para ingresar al motor de inferencia .

[b] Manejador de Base de Datos : suministra información sobre el estado inicial del problema a requerimiento del motor de inferencia .

[c] Generador de Distorsiones : establece que premisas deducidas por el motor de inferencia deben ser consideradas como falsas después de la interacción con el mundo real y determina los períodos de actualización del sistema (interacción con el mundo real) .

[d] Motor de Inferencia : es el mecanismo que se encarga de realizar las inferencias a partir de las reglas habilitadas y la información contenida en la Base de Datos y la Memoria de Trabajo .

[e] Manejador de Memoria de Trabajo : recibe del motor de inferencia la información generada por el proceso deductivo y le suministra información deducida por aquel a partir del estado inicial del problema .

4. Las Variables :

Las variables independientes definidas para el estudio son :

Cantidad de reglas del sistema
Se han tomado Sistemas desde 50 hasta 500 reglas .

Cantidad de reglas disparadas
Se han observado experimentos donde esta variable toma valores en la escala [0,1200]

Lapso de actualización
Se ha definido el lapso de actualización en función de la cantidad de reglas disparadas tomando valores en la escala [5,25] , el valor "sin actualización" equivale al sistema sin distorsiones .

Cantidad de datos que describen el problema
Se han tomado Bases de Datos desde 10 hasta 50 datos .

Rango de evidencia de datos :
Se han tomado los siguientes
[0.0,0.1] , [0.1,0.2] , [0.2,0.3] , [0.3,0.4] , [0.4,0.5] ,
[0.5,0.6] , [0.6,0.7] , [0.7,0.8] , [0.8,0.9] y [0.9,1.0].

Rango de evidencia de reglas :
Se han tomado los siguientes
[0.0,0.1] , [0.1,0.2] , [0.2,0.3] , [0.3,0.4] , [0.4,0.5] ,
[0.5,0.6] , [0.6,0.7] , [0.7,0.8] , [0.8,0.9] y [0.9,1.0].

Cantidad de ciclos :
Se ha tomado el rango [0,30] .

Porcentaje de Reglas con Evidencia Negativa :
Se ha tomado el rango [0%,100%] .

Se han tomado las siguientes variables dependientes :

Porcentaje Promedio de Reglas Disparadas :
Se ha tomado el rango [0,1400] .

Porcentaje Promedio de Nodos (hechos) Saturados :
Se ha tomado el rango [0,20] .

Porcentaje de Nodos Saturados :
Se ha tomado el rango [0,120] .

Promedio de Ciclos para Saturar un Nodo (hecho) :
Se ha tomado el rango [0,30] .

Nivel de Estabilización :
Se ha tomado el rango [0,30] .

Definimos el Punto de Degradación del Sistema a aquel punto en el que la cantidad de reglas disparadas supera la cantidad de reglas del Sistema .

5. Las Pruebas :

La experimentación permite comprender el funcionamiento de los SMNV a través de :

[a] El estudio de la variación del promedio de la cantidad de reglas disparadas hasta alcanzar el objetivo [11] en función de la cantidad de reglas del sistema y del lapso de actualización .

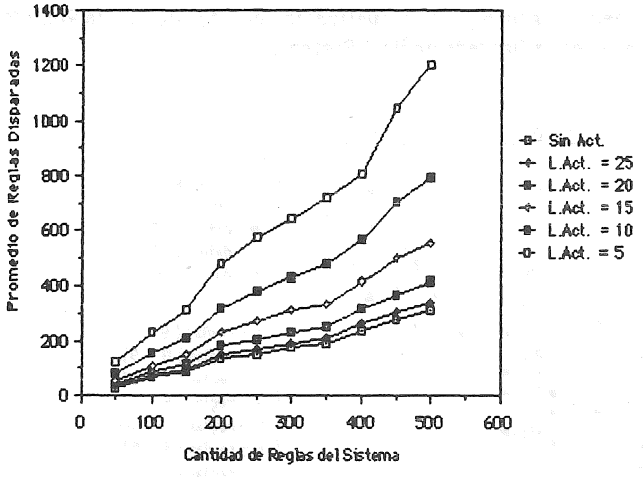
[b] El estudio de la variación del promedio de la cantidad de reglas disparadas hasta alcanzar el objetivo en función de la cantidad de datos necesarios para describir el problema y del lapso de actualización .

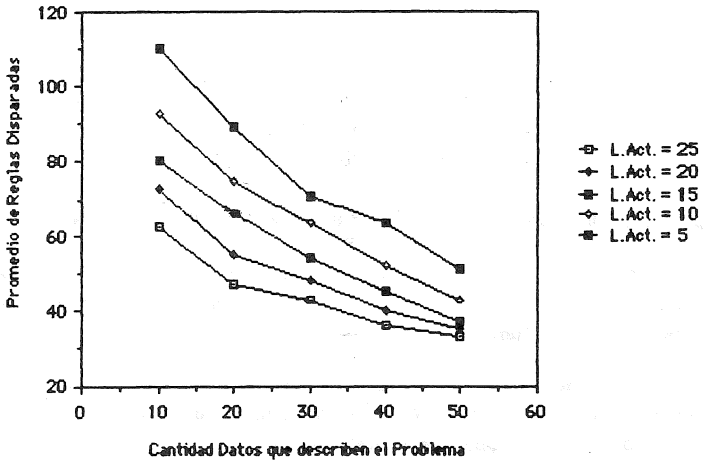
[c] El estudio de el porcentaje de nodos saturados [12] en función de la evidencia inicial asociada a cada uno de los hechos y la evidencia asociada a cada una de las reglas.

[d] El estudio de la incidencia de los rangos de evidencia de hechos y reglas en la saturación de nodos.

[e] El estudio de la incidencia de los rangos de evidencia de hechos y reglas en la cantidad de ciclos necesarios para saturar un nodo.

6. Gráficas :





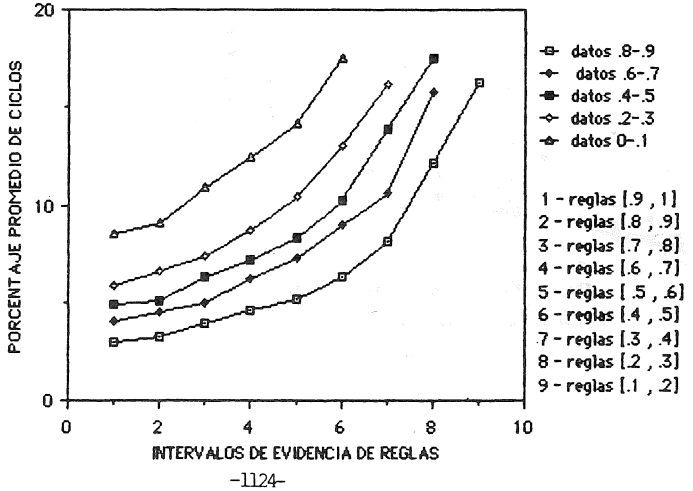
La primera gráfica que corresponde al primer experimento permite establecer el siguiente comportamiento :

COMPORTAMIENTO 1 : A mayor Masa de Conocimiento (cantidad de reglas), mayor es la Degradación del Sistema .

La segunda gráfica que corresponde al segundo experimento permite establecer los siguientes comportamientos :

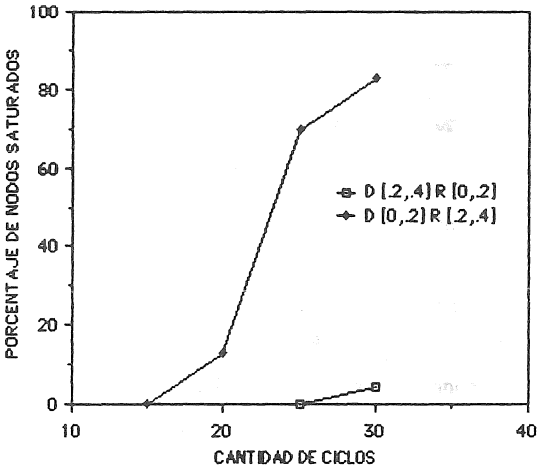
COMPORTAMIENTO 2 : A mejor definición del modelo (más datos en la Base de Datos) , menor es la Degradación del Sistema .

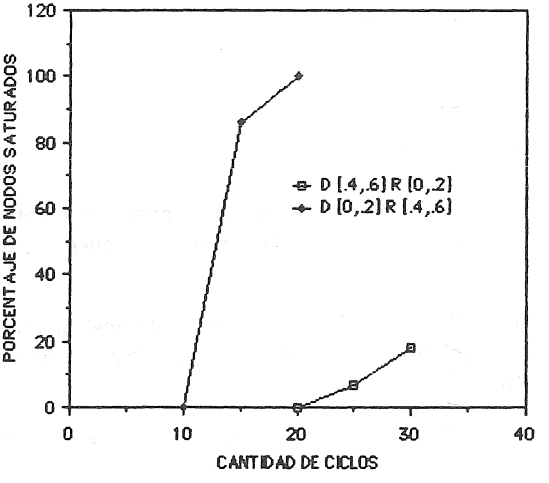
COMPORTAMIENTO 3 : A mayor frecuencia de actualización del Modelo con la Realidad, mayor es la Degradación del Sistema .

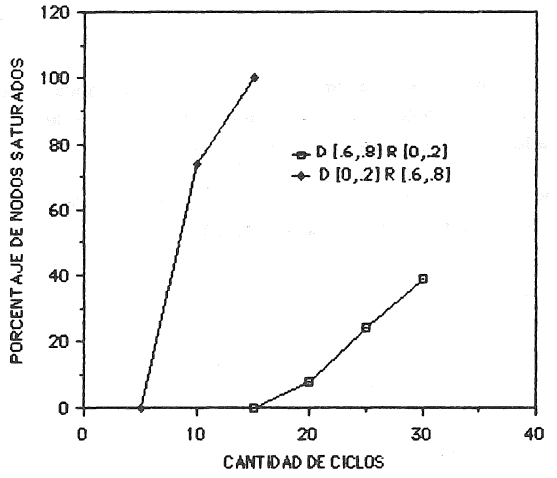


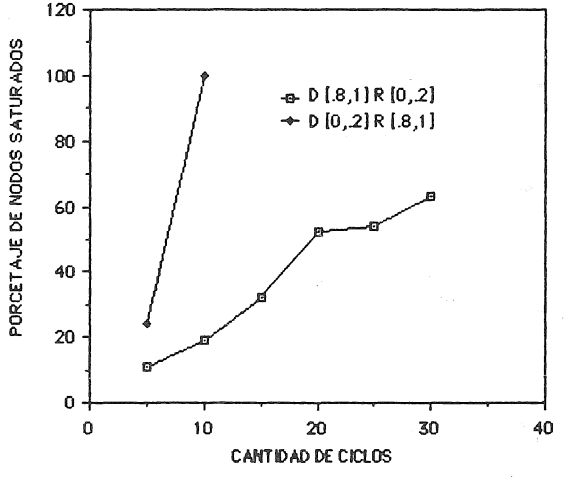
La gráfica precedente permite confirmar que la cantidad de ciclos para alcanzar la saturación de un nodo es inversamente proporcional al rango de evidencia de los datos y las reglas [14,15,16], esto puede expresarse mediante el siguiente comportamiento :

COMPORTAMIENTO 4 : A mayor evidencia de datos y reglas se necesita menor cantidad de ciclos para alcanzar la saturación.





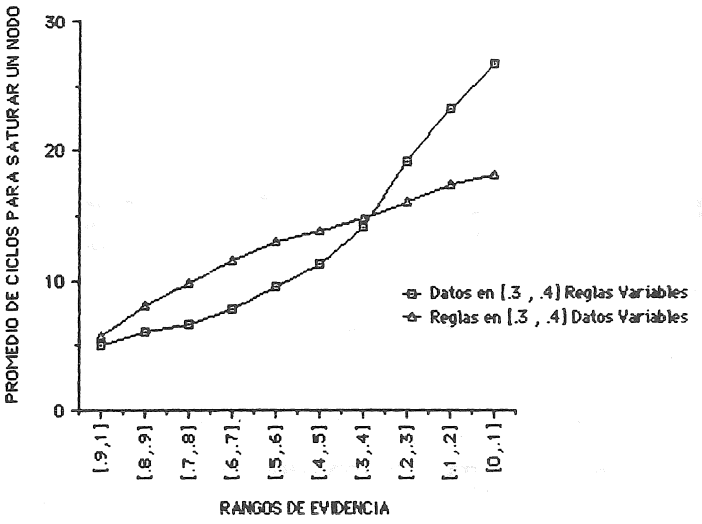


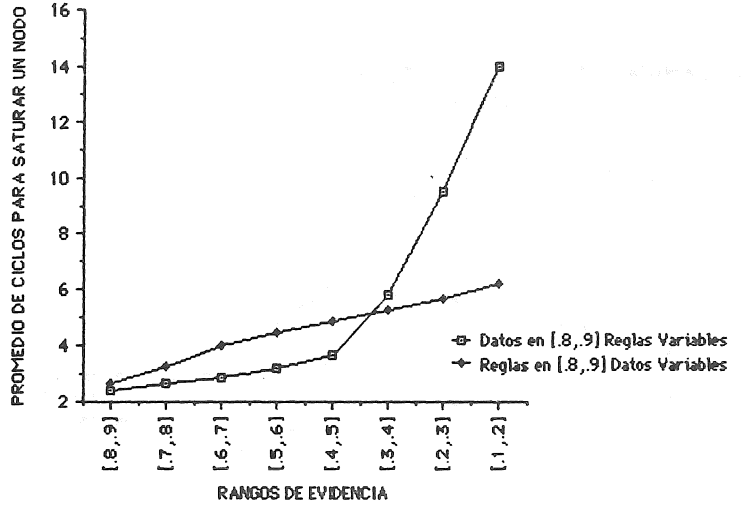


Las cuatro gráficas precedentes proveen información sobre el comportamiento de los nodos saturados en función de los ciclos a partir de las mismas se puede postular el siguiente comportamiento :

COMPORTAMIENTO 5 : Los nodos se saturan más rápido (menor cantidad de ciclos) con rango de evidencia de reglas alto que con rango de evidencia de datos alto.

Este comportamiento permite inferir que para el modelo importa más estar seguro de las reglas (rango de evidencia de reglas alto) que de los hechos (rango de evidencia de datos alto).

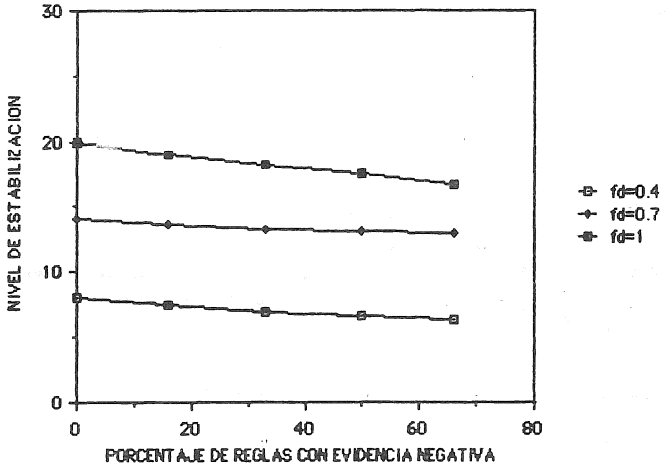




Las dos gráficas precedentes proveen información sobre el promedio de ciclos necesarios para saturar un nodo en función de los rangos de evidencia, a partir de las mismas se pueden postular los siguientes comportamientos :

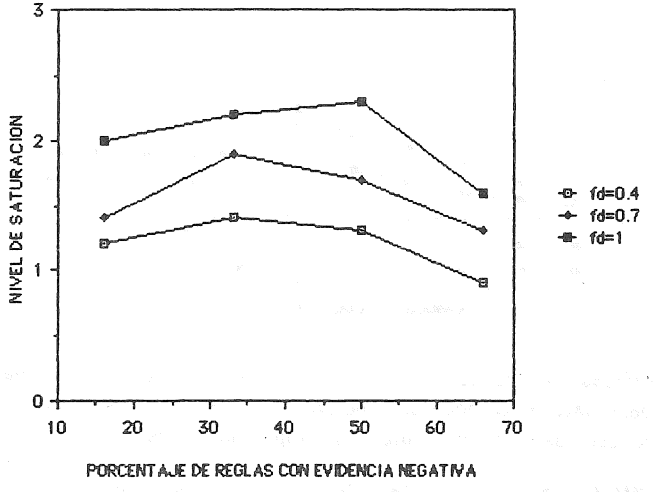
COMPORTAMIENTO 6 : El porcentaje de ciclos necesario para saturar un nodo crece linealmente con el punto medio del rango de evidencia de reglas.

COMPORTAMIENTO 7 : El porcentaje de ciclos necesario para saturar un nodo crece polinomialmente con el punto medio del rango de evidencia de datos.



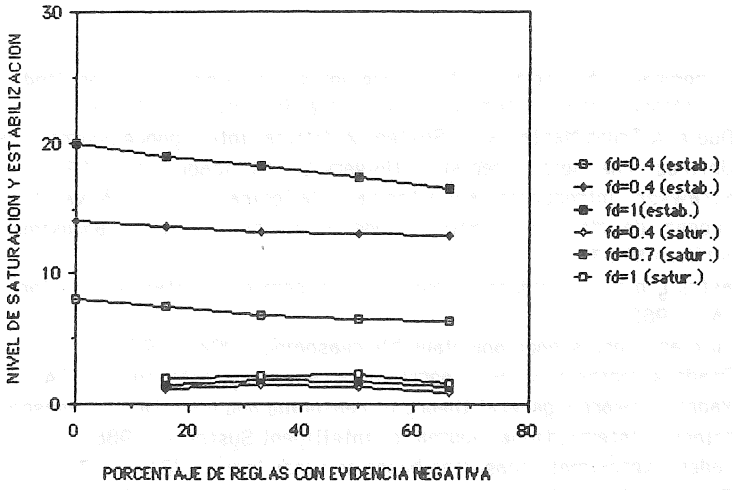
La gráfica precedente provee información sobre el nivel de estabilización alcanzado por el sistema en función del porcentaje de reglas con evidencia negativa, a partir de la misma se pueden postular el siguiente comportamiento :

COMPORTAMIENTO 8 : La estabilización en SNMV es directamente proporcional al factor de dispersión.



La gráfica precedente provee información sobre el nivel de saturación alcanzado por el sistema en función del porcentaje de reglas con evidencia negativa, a partir de la misma se pueden postular el siguiente comportamiento :

COMPORTAMIENTO 9 : La saturación en SNMV es directamente proporcional al factor de dispersión.



La gráfica precedente se construye mediante la combinación de las dos anteriores y provee información sobre la comparación entre el nivel de saturación y el nivel de estabilización alcanzado por el sistema en función del porcentaje de reglas con evidencia negativa, a partir de la misma se pueden postular el siguiente comportamiento :

COMPORTAMIENTO 10 : A igualdad de condiciones, el nivel de saturación en SNMV es mayor que el nivel de estabilización.

7. Conclusión :

La investigación en este campo recién comienza por lo que una conclusión es poco menos que aventurada , lo que se ha presentado son los primeros resultados de un proyecto de investigación llevado adelante en el Programa de Investigación y Desarrollo en Inteligencia Artificial de la Universidad Nacional de Luján que tiene como objetivo tecnológico diseñar un Entorno de Desarrollo de Sistemas Expertos basado en un sistema no monotónico de representación de la verdad . Esta tecnología [13,14,15] está marcando el rumbo de la próxima década en lo que hace a los avances de la manipulación del conocimiento. Los sistemas de mantenimiento de la verdad constituyen modelos adecuados para representar las variaciones de evidencia en los llamados universos de hipótesis, la investigación en esta área se hace necesaria si se quiere tener modelos de representación y razonamiento con un buen ajuste al mundo real.

8. Bibliografía :

[1] K. Colby. Simulations of Belief Systems , en : Computer models of Thought an Language. W.H. Freeman Publishing Co. USA . 1973

[2] J. Doyle. Explicit control of reasoning. Proceedings of the ACM Symposium on Artificial Intelligence and Programming Languages . Asociation of Computer Machinary . USA . 1977

[3] S. Fahlman. A System for Representing and Using Real World Knowledge MIT Press . USA . 1979

[4] A. Thompson. Network Truth Mantainance for Deduction and Modelling. Sixth International Joint Conference on Artificial Intelligence . 1979
[5] J. Doyle. A Truth Mantainance System. Artificial Intelligence Journal . USA . 1979
[6] R. Jeffrey . The logic of decision . University of Chicago Press . 1983 .
[7] F. Strawson . Introducción a una teoría de la lógica . Editorial Nova . 1982 .
[8] R. Lopez de Mantaras Badía . Modelos de razonamiento aproximado . Mundo científico Nº 53 .
[9] Shastri y otros . Evidential reasoning in semantic networks : a formal theory . IJCAI . 1985 .
[10] J. Quinlan . Consistency and plausible reasoning . IJCAI . 1979 .
[11] H. Prade. A sinthetic view of aproximate reasoning techniques . IJCAI . 1983 .
[12] R. Yager . Toward a general theory of reasoning and uncertainty : nonspecificity and fuzziness . Internartional Journal of intelligent Systems . 1986
[13] L. Zadeh . Aproximate reasoning based on fuzzy logic . IJCAI . 1979 .
[14] H. Prade y otros . On the unicity of Dempster rule of combination . Internartional Journal of intelligent Systems . 1986 .
[15] D. Driankov . Uncertainty calculus with verbally defined belief intervals . Internartional Journal of intelligent Systems . 1986 .

9. Contacto :

Ramón García Martínez
Erézcano 1134
Adrogué (1846)
Buenos Aires
ARGENTINA

SISTEMAS EVOLUTIVOS: Nuevo Paradigma de la Informática

Fernando Galindo Soria

Licenciatura en Ciencias de la Informatica

Seccion de Graduados e Investigacion

UPIICSA-IPN
Calle de Té # 950
Col. Granjas México
Delegación Iztacalco
08400 México, D.F.
579-94-22 Ext. 301 o 206

RESUMEN

En este documento se presenta una introducción a los Sistemas Evolutivos (Sistemas capaces de construir y mantener en tiempo real una imagen del ambiente que los rodea, como un ser que construye su propia imagen de la realidad y la utiliza para interactuar con su entorno).

Incluyendo la descripción general de un Sistema Evolutivo y presentación de algunos de los métodos y algoritmos más usados en la construcción de este tipo de sistemas principalmente la Arquitectura del Sistema Evolutivo y su estructura Lingüística y describiendo el Mecanismo Léxico, el Mecanismo Sintáctico , las principales herramientas de Inferencia Gramatical, el Mecanismo Semántico y las principales herramientas de Diálogo y de asociación de significado.

PALABRAS CLAVES

Sistemas Evolutivos, Lingüística Matemática, reconocimiento de Formas, Sistemas de Información, Base de Datos, Inferencia Gramatical, Herramientas Automátizadas.

SISTEMAS EVOLUTIVOS: NUEVO PARADIGMA DE LA INFORMATICA

I N T R O D U C C I O N

En éste documento se describe en forma general la arquitectura y el área de aplicación de los Sistemas Evolutivos, donde un Sistema Evolutivo es una herramienta automatizada capaz de construir una imagen del ambiente que la rodea y a partir de esta imagen resolver problemas, como un niño que construye su propia imagen de la realidad y la utiliza para interactuar con su entorno.

Como primer punto se describe en forma general, cuales son los principales problemas de la Informática actual y la arquitectura general de los Sistemas de Información y a partir de este punto se sigue una secuencia en espiral donde en cada vuelta de la secuencia se replantea la arquitectura de los sistemas de información hasta terminar en la conclusión con la arquitectura generalizada de los Sistemas Evolutivos.

1). Introducción a los Sistemas Evolutivos.

Uno de los problemas más graves de la Informática actual se presenta por su poca capacidad para modelar en tiempo real los fenómenos que ocurren en la realidad, ya que es comùn que cuando un sistema de Información (Nómina, Compilador, Reconocedor de Imágenes, Sistema de Inventarios, Sistema Experto de Diagnóstico Médico, etc.) es liberado ya prácticamente es obsoleto, ya sea porque:

a) El problema modelado se modificó.

b) Porque el modelo no cubrió los aspectos esenciales.

c) O simplemente la información y el conocimento que se tiene sobre el tema ha quedado rebasado por algún nuevo dato o hecho conocido previamente.

En general se pueden plantear tres grandes problemas que ayudan a la rápida obsolencia de los sistemas de información y que obligan a realizar en forma continua lo que se conoce como mantenimiento adaptativo (aquel mantenimiento que se realiza para lograr que el sistema mantenga una imagen lo más cercana al ambiente que se requiere modelar) o por el contrario obliga a los usuarios a lidiar con una herramienta cada vez menos poderosa y más alejada de la realidad que aparentemente está modelando.

El primer problema surge cuando se desarrollan sistemas que modelan fenómenos altamente cambiantes, como por ejemplo los sistemas de algo tan cotidiano como la Nómina, donde prácticamente no se ha terminado de desarrollar cuando ya se tiene que modificar (y no es por mal desarrollo, sino simplemente porqué se esta tratando de modelar un problema que cambia prácticamente por decreto y en el que es difícil establecer los patrones de cambio).

El segundo problema se presenta cuando se desarrollan sistemas que modelan fenómenos para los cuales no existe una regla o patrón ya establecido y bien manejado y del cual se pueda tener un algoritmo en forma relativamente sencilla, tal es el caso de muchos de los problemas actuales de la Inteligencia Artificial, por ejemplo en el reconocimento de patrones se ha desarrollado gran cantidad de métodos y algoritmos pero sin embargo los problemas no resueltos son cada día mayores, por lo que los Sistemas de Información orientados al reconocimiento de patrones reflejan un modelo de la realidad normalmente restringido.

El tercer problema es un cuello de botella de la Informática que se presenta cuando los sistemas funcionando son incapaces de reflejar la realidad por que los datos y hechos de ésta no son directamente accesibles por el sistema ya sea porque son difíciles de obtener (por ejemplo: Topografía detallada del terreno donde se construirá una carretera) o porqué cambian tan rapidamente que los métodos tradicionales de captura no permiten mantenerlos actualizados (por ejemplo los datos clínicos de un paciente de terapia intensiva).

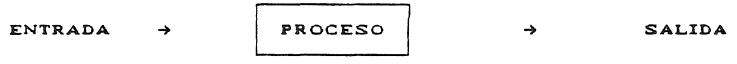
Es por lo anterior que es necesario replantear radicalmente el enfoque utilizando para resolver problemas en Informática (representado por áreas como el Desarrollo de Sistemas ,la Ingeniería de Software y la Ingeniería de Conocimiento) en el cual la tendencia es a la construcción de sistemas Estaticos e Incapaces de automantenerse y buscar métodos y herramientas capaces de recrear en forma continua su imagen de la realidad o del problema a resolver.

Es dentro de este contexto donde surge el concepto de **Sistema evolutivo** como un sistema capaz de crear su propia imagen de la realidad y utilizarla para resolver problemas y en su momento mantener actualizada esta imagen mediante un proceso continuo de actualización.

2) Del sistema de Información al Sistema Evolutivo.

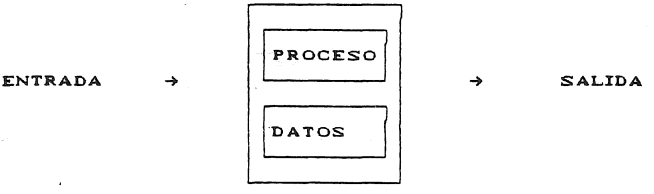
2.1) Rumbo al Constructor

En general se considera que un Sistema de Información o programa de cómputo tiene la siguiente arquitectura.

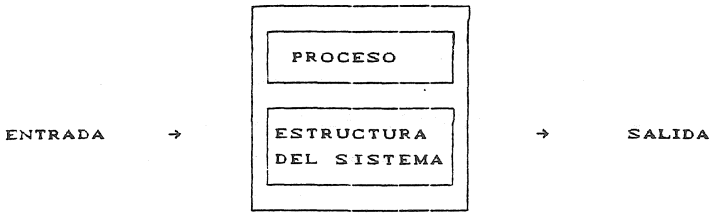


Sin embargo ,este esquema es muy viejo y corresponde a los años 50's y ya para la década de los 60's se manejaban los siguientes esquemas más elaborados:

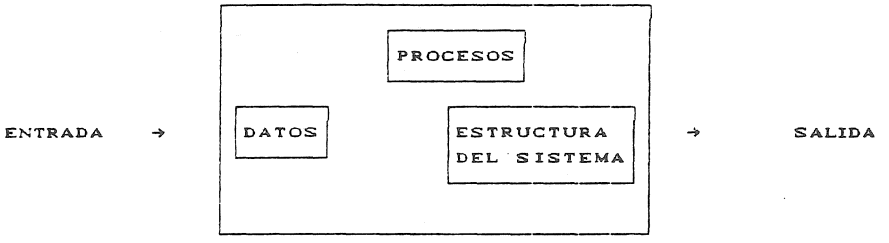
a)



b)

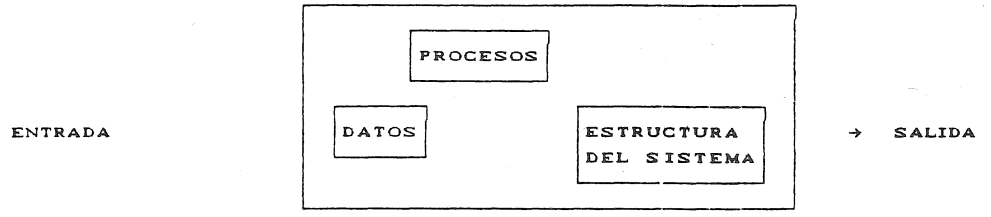


El primer esquema se utilizaba principalmente por la gente orientada al desarrollo de aplicaciones y el segundo por las personas dedicadas a desarrollar herramientas de cómputo como Compiladores y Sistemas Operativos, y ya para finales de los 70's se manejaba un modelo generalizado en el cual se considera que cualquier sistema de información tiene la siguiente arquitectura.



Este esquema integra los enfoques anteriores y enfatiza la necesidad de desarrollar los sistemas tomando en cuenta las tres componentes y no solo los procesos, sin embargo, el desconocimiento o el hecho de no tomar en cuenta este esquema cuando se desarrollan sistemas es una de las causas principales por las que los sistemas se vuelven altamente estáticos y difíciles de mantener ya que en los sistemas y programas tradicionales las tres componentes se encuentran revueltas, por lo que, un cambio "pequeño" en los datos o procesos o en el orden de atacar un problema ocasiona que practicamente se tenga que volver a programar todo; por otro lado si se desarrolla el sistema de tal manera que los datos queden en un lado, los procesos en otro y finalmente la estructura del sistema en otro, el proceso de actualización puede ser relativamente fácil (un caso particular de este enfoque es el del desarrollo de Bases de Datos).

Por lo que se considera que una característica fundamental que se debe buscar cuando se desarrolla un sistema de información es la de que exista una INDEPENDENCIA RELATIVA o sea que los datos, procesos y estructura del sistema queden separados y únicamente exista la relación minima necesaria entre las tres componentes.

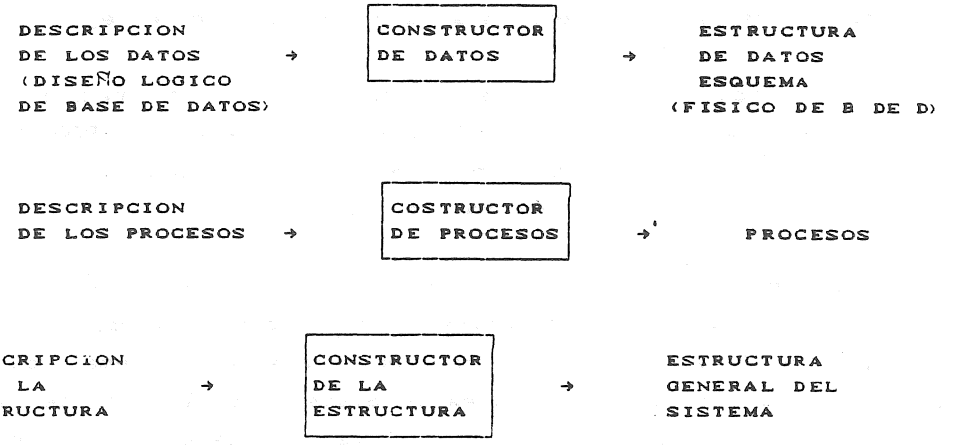


La Independencia Relativa es una de las características distintiva del método de desarrollo basado en Base de Datos y en este documento simplemente se extiende al esquema general de Sistemas de Información.

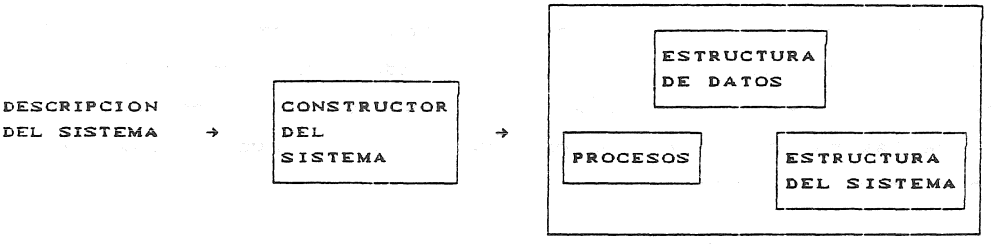
Otra característica presente en el método de Base de Datos que se puede extender al desarrollo de Sistemas es el Hecho de que en el enfoque de Base de Datos se distinguen explicitamente tres niveles: en el primero se encuentran los datos en sí (nombres propios, edades concretas, direcciones especificas, etc.); en el segundo se tiene la estructura de datos especifica donde se almacenan los datos (Archivos, registros y campos de la Base de Datos que se esta manejando) y en el tercer nivel se tiene un programa que se encarga de construir una estructura de datos particular, a partir de una descripción general de la Base de Datos, este programa es el Constructor de la Base de Datos.

En realidad una base de datos no tiene al principio datos o estructuras de datos particulares, sino que solo cuenta con el constructor que es capaz de generar multiples casos particulares.

En el caso de sistemas de información esta idea se puede generalizar, buscando que, más que tener datos, procesos o estructuras particulares de un sistema, se cuente con mecanismos que nos permitan construir los datos, procesos y estructuras a partir de una descripción general del sistema, como se muestra en el siguiente esquema.



Estos tres módulos se pueden integrar en el siguiente esquema general.



Donde el constructor del sistema es un programa capaz de construir la estructura general de un Sistema de Información a partir de la descripción del sistema, de la misma forma que un constructor de Base de Datos es capaz de construir el esquema físico de una Base de Datos a partir de un Esquema Lógico de Base de Datos.

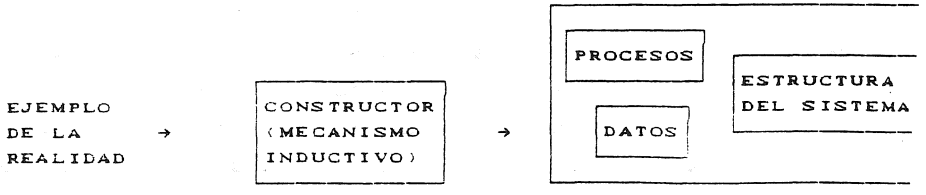
3.2) Rumbo a la Arquitectura de Sistema Evolutivo

Si se construye el sistema de información de tal forma que los componentes sean independientes en forma relativa entre sí y se da una interrelación entre el Constructor y el Sistema de tal forma que cualquier cambio en la Descripción del Sistema se refleje en tiempo real en el Sistema de Información, entonces se puede considerar que la imagen de la realidad que tiene el sistema de información es bastante cercana a la que se quiere reflejar.

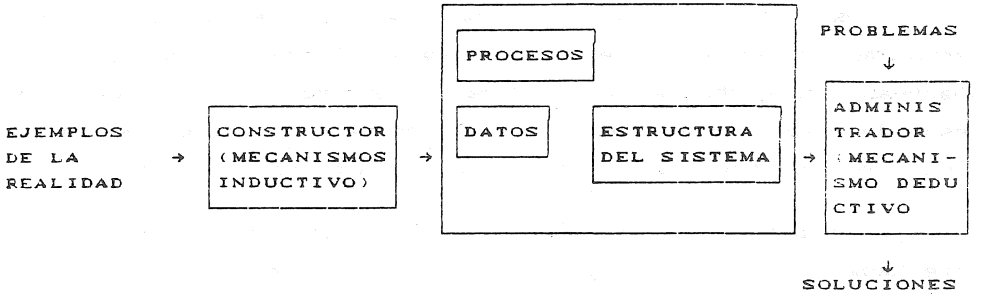
Ahora bien cuando un programa sigue un conjunto de reglas o instrucciones para resolver un problema se dice que es un PROGRAMA DEDUCTIVO, y a los métodos que construyen programas deductivos se les conoce como métodos de programación deductiva.

El problema de la programación deductiva es que se orienta ha desarrollar sistemas fijos difíciles de modificar en tiempo real (ya que las modificaciones involucran reprogramar el Sistema), por lo que desde hace tiempo en el área de Sistemas Evolutivos se ha buscado desarrollar herramientas automátizadas capaces de obtener en forma automática el conjunto de reglas del sistema a partir de ejemplos y descripciones generales de un programa.

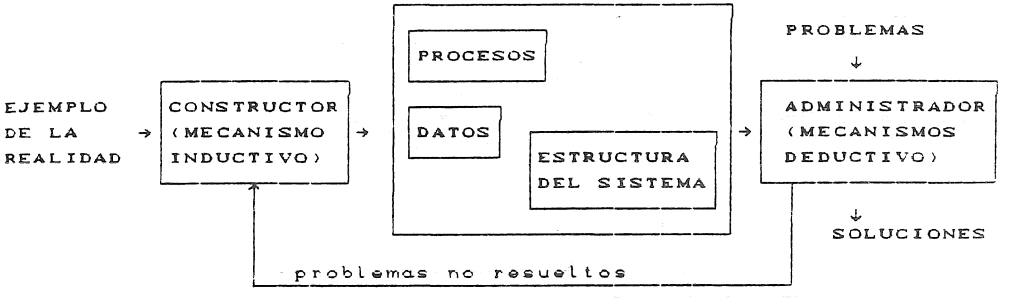
A una herramienta o programa capaz de encontrar un conjunto de reglas a partir de ejemplos la conocemos como MECANISMO O HERRAMIENTA INDUCTIVA; los mecanismos inductivos fueron de las primeras herramientas utilizadas en el desarrollo de constructores y son de las más generales, el esquema de un constructor basado en una herramienta inductiva es:



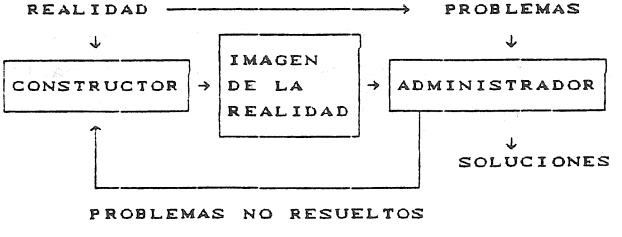
En el esquema anterior se maneja un mecanismo inductivo para construir una Imagen de la Realidad, sin embargo en esta imagen solo se tienen un conjunto de reglas en términos de procesos, Estructura y Datos del Sistema por lo que es necesario integrar una herramienta capaz de resolver problemas siguiendo esas reglas, o sea un mecanismo de tipo deductivo como se ve en el siguiente diagrama.



Para contar con un SISTEMA EVOLUTIVO lo único que hace falta es dotar a esta herramienta con la capacidad de actualizar su imagen de la realidad en tiempo real con lo que se tiene la siguiente arquitectura :



De donde la Arquitectura General de un Sistema Evolutivo es.



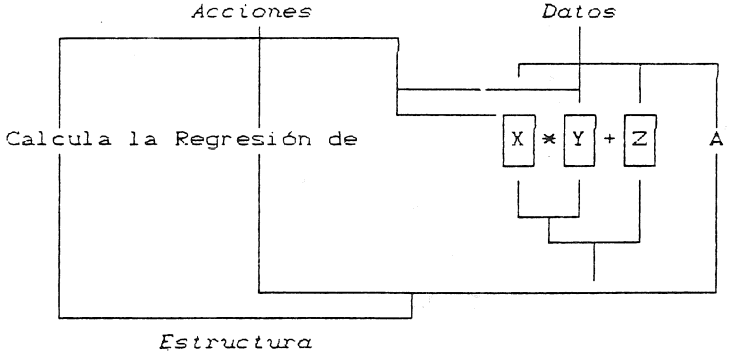
3) ARQUITECTURA

En el punto anterior se vió que con el fin de resolver problemas el sistema evolutivo construye una imagen de la realidad, pero no se indicó como se lleva a cabo este proceso, por lo que en este punto se describirá en forma general los principales métodos y algoritmos utilizados por los Sistemas Evolutivos; para lo cual se partirá de que **el lenguaje de comunicación con el exterior (o sea el lenguaje con el que se plantean los problemas al Sistema Evolutivo y con el cual éste responde a los requerimientos) y el lenguaje con el que el Sistema Evolutivo construye su imagen de la realidad son el mismo.**

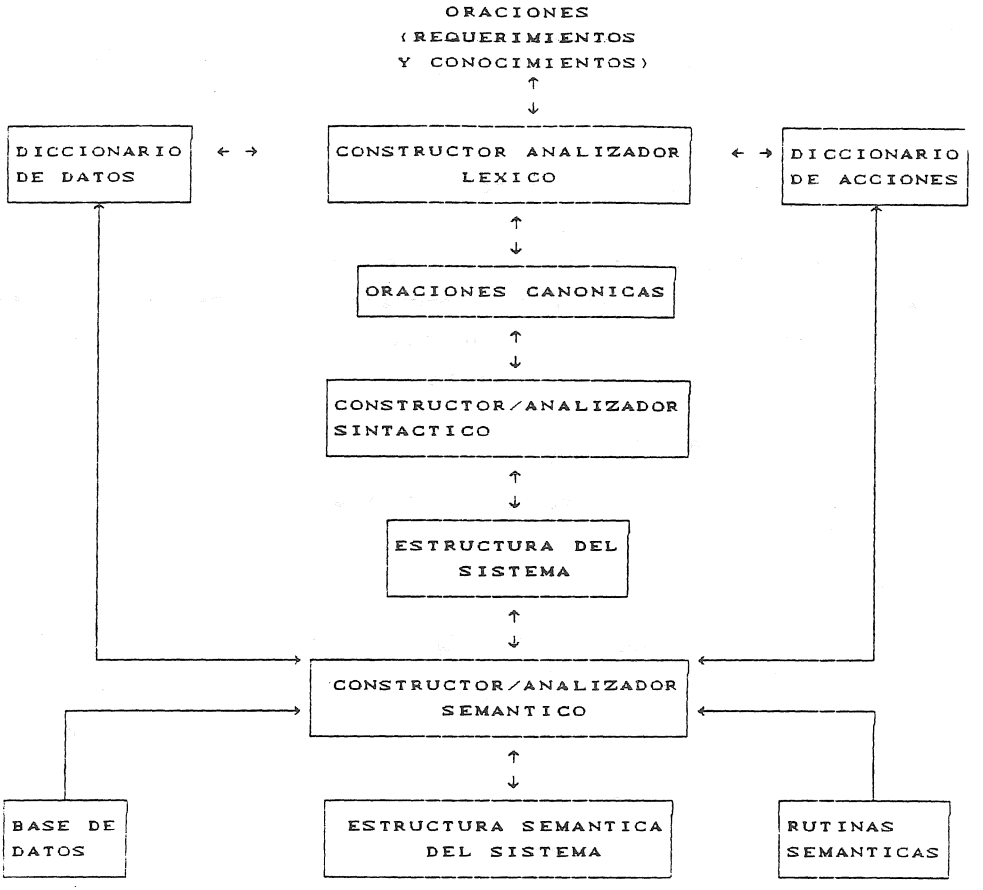
En la actualidad ya existen una gran cántidad de herramientas orientadas al manejo lingüístico de los Sistemas Evolutivos y se basan en que es relativamente fácil encontrar los elementos de un Sistemas dentro de las oraciones, por ejemplo en la Oración:

Calcula la Regresión de X * Y + Z

Es relativamente fácil detectar los datos, acciones y estructura, como se ve a continuación.



Entonces un Sistema Evolutivo tiene que tener la capacidad de encontrar los componentes del sistema de información a partir del lenguaje utilizado en el área problema ,y para lograrlo muchos sistemas evolutivos tienen la siguiente arquitectura:



Donde el Constructor/analizador Léxico es el encargado de encontrar cada una de las unidades lexicas que componen una oración, de que tipo es cada unidad (por ejemplo dato ú oración) y la estructura general de la oración conocida como oración Canónica (formada por la concatenación de los tipos de las unidades lexicas).

A partir de ahi el Constructor/Analizador Sintáctico aplicando metodos de inferencia Gramatical (Métodos orientados a encontrar la gramática de un lenguaje a partir de ejemplos de oraciones del lenguaje) encuentra la estructura del sistema y finalmente el Constructor/Analizador Semántico, encuentra significado de la oración utilizando mecanismos de dialogo para encontrar el significado de los elementos desconocidos.

A continuación se describiran algunos de los principales métodos utilizados por cada uno de los constructores/analizadores.

3.1) Constructor/Analizador léxico

El Constructor/Analizador Léxico es un programa que recibe como entrada una oración en algún lenguaje y obtiene las unidades lexicas presentes en la oración, el tipo de cada unidad lexica y la estructura general u oración Canónica.

Por ejemplo, dada la Oración

Calcula la Regresión de X * Y , Z

se encuentra

TABLA DE UNIDADES LEXICAS

Unidad Lexica	*Tipo de Unidad Lexica*	*Oración Canonica* a i a i d a d a d
Calcula	Acción (a)	
la	Ignorada (i)	
Regresión	Acción (a)	
de	Ignorada (i)	
X	Dato (d)	
*	Acción (a)	
Y	Dato (d)	
,	Acción (a)	
Z	Dato (d)	

Y asi para cada oración.

Originalmente el Constructor/Analizador Léxico no tiene ningún conocimiento acerca del lenguaje a utilizar y solamente cuenta con la capacidad para encontrar los diferentes tipos de unidades lexicas de un lenguaje para lo cual utiliza entre otros los siguientes métodos "tribiales":

Unidades Predefinidas
Caracteres Especiales
Manejo de Contexto
Mecanismo de Dialogo

a) El uso de Unidades Predefinidas es el método más usado y el más antiguo y consiste básicamente en almacenar previamente la tabla de unidades lexicas con las palabras que se utilizan en el sistema e indicando de que tipo son. Con la caracteristica de que en un sistema evolutivo esta tabla es creciente (O sea que originalmente la tabla tiene las unidades lexicas mas usadas en el sistema y se cuenta con la capacidad de almacenar nuevas unidades conforme se van detectando).

b) Caracteres especiales

El mecanismo anterior tiene la limitante de que si se requiere utilizar una palabra no catalogada el sistema tiene que preguntar y ésto al principio puede ser tedioso ,por lo que ,cuando se construyeron los primeros ensambladores y compiladores se utilizó el truco de manejar ciertos caracteres o posiciones especificas dentro del programa para que el sistema asignara las unidades lexicas.

Por ejemplo en un Ensamblador dado, la instrucción:

```
:Etiqueta   .Mem    #3    ;Coloca 3 en la etiqueta
```

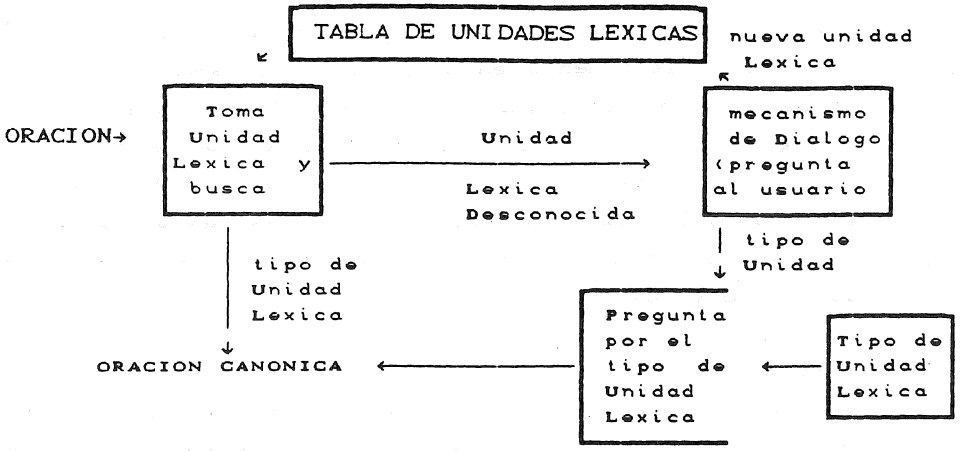
es facilmente analizable si se tiene la convención de que:

:	Indica Tipo de Etiqueta
.	Indica Tipo de Instrucción
#	Indica tipo de Dato
;	Indica Tipo de Comentario

En general esta idea se puede utilizar para catalogar nuevas unidades Léxicas.

c) Mecanismo de Díalogo

En un Sistema Evolutivo Muchas veces se desconoce hasta el tipo de unidades léxicas que conforman el lenguaje, por lo que, se ha visto que un mecanismo muy general es aquel que permite catalogar en tiempo real las nuevas unidades Léxicas, para lo cual se maneja el siguiente esquema :



Este mecanismo "Tribial" es extremadamente fuerte y refleja el proceso que sigue un niño cuando desconoce una palabra.

d) Manejo del Contexto

Los Anteriores métodos tienen el inconveniente de que todo se le tiene que dar al sistema y no utilizan mucha de la información ya obtenida. En el momento que ya se han manejado varias oraciones se pueden empezar a detectar algunos patrones que se pueden utilizar para encontrar el tipo de nuevas unidades léxicas.

Por ejemplo; si después de catalogar 10 oraciones se encontró que la estructura

a i a i d

se repite 4 veces y el sistema recibe la oración :

Calcula la media de A

donde ya conoce el tipo de las unidades :

calcula (a)
la (i)
de (i)
A (d)

entonces la oración canónica tiene la forma

a i ? i d

Por lo que el sistema propone que el elemento desconocido MEDIA es de tipo a.

Los anteriores métodos "tribiales" al convinarse son extremadamente poderosos y en su momento con una cantidad relativamente pequeña de ejemplos son capaces de encontrar el conjunto base de palabras manejadas por un área problema.

Existen otros métodos más poderosos pero normalmente requieren del apoyo de los mecanismos Sintáctico y Semántico, y normalmente no se utilizan en los prototipos de Sistemas Evolutivos.

3.2) Constructor/Analizador Sintáctico.

A partir de los resultados del Constructor/Analizador Léxico el Constructor/Analizador Sintáctico encuentra la estructura del sistema, para lo cual, toma como entrada el conjunto de oraciones canónicas y genera la Estructura.

Este mecanismo es tal vez el componente mas importante de un sistema Evolutivo ya que es el responsable de encontrar las reglas generales o patrones de estructura del sistema y para lograrlo utiliza normalmente métodos de la inferencia gramatical.

La Inferencia Gramatical es una herramienta de la lingüística Matemática utilizada originalmente en el área de reconocimiento de Patrones y que posteriormente se ha extendido y usado masivamente en la construcción de Sistemas Evolutivos.

El problema que ataca la Inferencia Gramatical consiste básicamente en encontrar la Gramática (Estructura) que describe a un lenguaje dado a partir de ejemplos de oraciones del Lenguaje.

En sus inicios se desarrollaron un conjunto de algoritmos orientados a resolver problemas específicos y en la mayoría de los casos eran métodos difíciles de entender y mas difíciles de programar, pero conforme se empezó a atacar el problema para contruir Sistemas Evolutivos se fueron encontrando nuevos métodos y generalizando el problema, por lo que, en la actualidad se ha encontrado que practicamente los métodos de inferencia gramatical se basan en las operaciones básicas de:

Factorización , Distribucion y Recursividad por lo que a continuación se explicaran estas operaciones.

a) Factorización Lingüística

En su Forma mas sencilla la factorización lingüística lo que pretende precisamente es encontrar los factores comunes en diferentes oraciones.

Por ejemplo si se tienen las siguientes oraciones canónicas (S índica el primer nivel de factorización).

S → a i b i a d o f
a i b i d o f
a i d a d f

factorizando la cadena a i que es común a las tres oraciones de S

S → a i X
X → b i a d o f
b i d o f
d a d f

Factorizando **b i** en las dos primeras oraciones de **X** queda

S → a i X
X → b i Y
d a d f
Y → a d o f
d o f

Si se observa, lo único que hace el proceso de factorización es sacar los factores comunes (que se repiten) de las diferentes oraciones.

b) Recursividad Lingüística

El proceso de recursividad lingüística también busca un conjunto de elementos comunes dentro de las oraciones pero con la diferencia de que se buscan cadenas de elementos que se repiten periodicamente y en forma consecutiva mas de cierto número mínimo de veces (normalmenete tres o más veces) y se asume que esa cadena se puede repetir tantas veces como se quiera.

Por ejemplo en la cadena

S a b c b c b c b c f

Los elementos b c se repiten consecutivamente 4 veces por lo que se asume que se pueden repetir tantas veces como se desee y esto se expresa introduciendo un elemento auxiliar y haciéndolo recursivo.

S → a X c e f

X → b c b c b c b c

↓

S → a X c e f

X → b c x

Por ejemplo si se tiene

S → a i a d a d a d a d f

Introduciendo recursividad

Sobre a d queda

↓

S → a i X f

X → a d X

El proceso de introducir recursividad es un mecanismo extremadamente poderoso ya que permite generalizar una secuencia de repetición y por el otro lado es un mecanismo peligroso ya que se puede generalizar mas de lo debido.

En general, el Constructor/Analizador obtiene la estructura de un sistema a partir de las oraciones del lenguaje y utilizando en forma combinada la recursividad distribución y factorización.

3.3) Constructor/Analizador Semántico

Esta componente del Sistema Evolutivo es la encargada de asociar el significado a los elementos y estructura del sistema y se basa principalmente en un mecanismo de dialogo y en la contrucción de relaciones entre los elementos utilizando diferentes mecanismos de captación de la realidad (En la actualidad utiliza principalmente imagenes y lenguaje escrito).

El mecanismo de Diálogo fué originalmente propuesto por José Luis Diaz Salaz en 1987 y posteriormente a sido complementado por otros investigadores y consiste basicamente en un Proceso Recursivo en el cual cada que el sistema Evolutivo detecta una palabra u oración para la que no encuentra significado, simplemente pregunta y espera una respuesta en el mismo lenguaje.

Por ejemplo si se tiene la oración

calcula la Media de A

y la computadora no sabe lo que significa la palabra MEDIA, pregunta y se puede establecer un dialogo del siguiente tipo:

Usuario:	Calcula la media de A
Computadora:	Cómo se obtiene la Media ?
Usuario	Suma los elementos de A y divídelos entre el número de elementos

. .
. .
. .

Este mecanismo ha sido modificado con el fin de permitir que se definan Sinónimos o unos elementos en término de otros

CONCLUSION

El área de los Sistemas Evolutivos es muy extensa y en este documento sólo se presentó una introducción al tema ya que desde 1985 se han desarrollado gran cantidad de sistemas de este tipo, principalmente a nivel de prototipos, pero ya en la actualidad varias empresas y entidades públicas están contruyendo sus propios Sistemas Evolutivos y al menos en un caso se está trabajando sobre una herramienta de mercado, por lo que ésta es una invitación a que más investigadores e instituciones se sumen a este esfuerzo.

- Algunas Publicaciones relacionadas con Sistemas Evolutivos

. Sistemas Evolutivos

  Construcción de Sistemas Evolutivos

. De la Biblioteca a la Memoria Automatizada

. Representación de Conocimientos

. Programación Dirigida por Sintaxis

. Inferencia Gramatical

. Introducción a la Lingüística Matemática

. Sistema Evolutivo Constructor de Sistemas Expertos

. Sistemas Evolutivos de Lenguaje de Trayectorias

. Herramientas Generales Para el desarrollo de juegos y Software Educativo

. Herramientas Generales para el Desarrollo de Sistemas (Núcleo de un Sistema Evolutivo).

MODELO PARA LA EVALUACION DE RENDIMIENTO DE SISTEMAS MANEJADORES DE BASE DE DATOS DISTRIBUIDAS[1]

JUDITH BARRIOS ALBORNOZ
ESCUELA DE INGENIERIA DE SISTEMAS
UNIVERSIDAD DE LOS ANDES
MERIDA. VENEZUELA

RESUMEN

Se presenta una vision intermedia para la evaluacion del rendimiento de los Sistemas Manejadores de Base de Datos Distribuidas (SMBDD). Para realizarla se define un modelo conceptual de rendimiento que permite el establecimiento de las relaciones entre los componentes internos del sistema distribuído tales como traductores y procesadores de comandos de usuario, procesadores locales de datos y, los componentes externos a este: aquellos que apoyan o soportan las actividades del sistema de base de datos, pero que no forman parte integral del mismo, estos son: el sistema de comunicaciones y transmisión de datos y el equipo computacional. La medida de rendimiento evaluada es el tiempo total de respuesta.

El estudio no es exhaustivo, sin embargo, permite observar las relaciones cambiantes entre los componentes del SMBDD cuando es sometido a la influencia de las aplicaciones que manipulan la base de datos en un ambiente distribuido.

Se designa como factor crítico de rendimiento a la visión distribuída de la base de datos y a las características particulares de cada una de las aplicaciones que accesan la base de datos distribuida. En este caso, su importancia no radica en el tiempo tomado para la definición de estrategia distribuída, sino en el tiempo total ahorrado por una buena decisión en la forma procesamiento de las solicitudes de los usuarios demandadas por las aplicaciones particulares.

1-INTRODUCCION

Las bases teóricas sobre las cuales se apoya un estudio de rendimiento, permiten establecer el contexto estudiado, las variables implicadas en el proceso, las restricciones y limitaciones tomadas en cuenta en la realización

[1] Trabajo realizado como Tesis de Maestria en Sistemas de Informacion en el Instituto Tecnologico y de Estudios Superiores de Monterrey (ITESM), Mexico 1990.

de la evaluación. La explicación del contexto, es definida en términos de modelos que reflejan la realidad, los cuales, pueden ser matemáticos, teóricos, de simulación o, de cualquier otro tipo que permita mostrar lo que ocurre en el sistema.

En el área de sistemas distribuidos, de acuerdo a la investigación bibliográfica realizada, no se ha desarrollado hasta el momento, ningún modelo general de rendimiento que permita explicar el comportamiento global de los sistemas manejadores de base de datos. Esto se debe, especialmente, a que los estudios de rendimiento sobre SMBDD's, han tenido dos orientaciones principales, ubicadas en los extremos de especialización y generalización de la evaluación.

La primera orientación, en la cual se ha concentrado la mayoría de las investigaciones, se avoca a la medición cuantitativa de tiempos de respuesta y de ejecución en modelos muy especializados. Dichos modelos explican comportamientos particulares y toman en cuenta, solamente, algunos de los componentes internos o de apoyo del sistema distribuído. Este enfoque es muy técnico y puntualiza los detalles de comportamiento de elementos críticos pertenecientes a la estructura interna del SMBDD. Entre algunas de las referencias que tienen este enfoque especializado estan [Bhar 89], [Bodo 88], [Casa 89], [Garc 88]. En este tipo de estudios, no se toma en cuenta la posibilidad de un enfoque más amplio; es decir, visualizar el contexto en el cual está inmerso el sistema distribuído. Al perder el concepto de globalidad, por la misma especialización, se dejan de incluir en el estudio elementos que contribuyen fuertemente en el rendimiento total del sistema. Sin embargo, no debe olvidarse que a medida que en un estudio se amplía el marco de referencia, se va perdiendo la oportunidad de profundizar en los detalles componentes de los elementos del sistema.

El segundo enfoque, opuesto al primero, es aquel en el que no se consideran los componentes especializados del sistema; sino que toma al sistema como una caja negra, la cual se somete a diferentes valores en sus factores de entrada y, se mide la respuesta o la velocidad de reacción del sistema ante los comandos introducidos. Esta salida es vista como tiempos de procesamiento y de respuesta, así como características de interfaz que el sistema evaluado proporciona al usuario. Algunos estudios incluyen las facilidades externas y las habilidades que ofrece el sistema para una

interacción hombre/máquina más amigable.(miden las cualidades de la interfaz proporcionada al usuario para facilitar su trabajo de desarrollo de aplicaciones) [Robi 88], [Youn 88], [Nich 88]. Las últimas referencias señalan evaluaciones para sistemas centralizados. Dado que el área de sistemas distribuídos es joven, las evaluaciones realizadas siguen la primera orientación (especializada) y los SMBDD's que existen, se encuentran aun en la fase experimental.

Tomando en consideración las referencias anteriores, en este proyecto se propone la definición de un modelo general de rendimiento para SMBDD. Dicho modelo señala los parámetros determinantes del comportamiento global de los SMBDD y las relaciones existentes entre ellos, colocando así esta investigación, en un punto intermedio entre los dos enfoques mencionados previamente.

Dado que el parámetro central de la evaluación, es el **tiempo total de respuesta**, la medición comienza con la introducción de una solicitud de información por parte del usuario del sistema y finaliza cuando se despliega dicha información al usuario. Por medio del modelo establecido, se analizan los cambios potenciales sobre la respuesta del sistema, al variar los valores de elementos externos al SMBDD (la frecuencia de consulta, el volúmen de datos implicados, el tipo de requerimientos del usuario, etc.). El cambio en el tiempo de respuesta, es causado por el grado en que varían las relaciones internas del SMBDD cuando se enfrentan a valores diferentes de la influencia del medio ambiente.

Para llevar a cabo este estudio, se toman como base fundamental, algunas de las investigaciones especializadas en el área de SMBDD. Dichas investigaciones proporcionan información sobre las características de comportamiento aislado de los diversos componentes del sistema distribuído específicamente en el nivel especializado. A traves de esta información, se definen tanto las características de los componentes del SMBDD, como las del conjunto de elementos externos sobre los cuales se apoya dicho sistema manejador para poder funcionar.

2- CONCEPTOS BASICOS sobre SISTEMAS MANEJADORES DE BASE DE DATOS DISTRIBUIDAS

La arquitectura de un SMBDD permite entender la configuración e interfaz entre los componentes del sistema, dando la posibilidad de convertir datos y programas de un formato a otro, haciendo más flexible la comunicación entre los distintos nodos del sistema distribuido. La arquitectura de referencia seleccionada como base del estudio [Lars 85a], maneja dos tipos de componentes: un conjunto de procesadores y un conjunto de esquemas. Los procesadores son partes de programas -software- ejecutables que tienen la responsabilidad de realizar una o más tareas. Pueden tener subprocesadores asociados para realizar algunas de las tareas más especializadas. Los esquemas son descripciones de algún aspecto lógico o físico de la base de datos. Son utilizados para que los procesadores realicen sus tareas eficientemente. Por lo tanto, un sistema de base de datos es el resultado de la cooperación entre diferentes procesos que siguen ciertos esquemas y manipulan un conjunto de datos locales. Extendiendo este concepto a sistemas distribuidos, los procesadores(usuario-datos) y los esquemas (externo, conceptual y de distribucion) ya no interactúan a nivel local, sino que se comunican por medio de una red de transmision de datos con otros procesadores y esquemas localizados en otros nodos; ampliando así, el marco de procesamiento para las diferentes tareas implicadas en una solicitud de usuario. La complejidad de cooperación esta oculta bajo un sistema de niveles y las interfaces entre ellos se establecen a través de un conjunto de reglas y protocolos, los cuales instruyen al elemento de nivel inferior sobre como estructurar los mensajes hacia los niveles superiores.

Ahora bien, teniendo en cuenta que el tipo de relaciones de cooperación e intercambio que existen entre los componentes del SMBDD pueden variar, es necesario, clasificar a los SMBDD en dos grandes grupos: los sistemas homogéneos y los sistemas heterogéneos:

1) Sistemas homogéneos presentan cierto grado de similitud entre sus componentes. De esta manera se hace más fácil el establecimiento del grado de interacción existente entre los elementos del modelo cuando se ejecutan las actividades de procesamiento distribuído. Las diferencias entre las características de implantación de cada uno de ellos, así como las diferencias

de equipo y de programas de apoyo que los soportan, no son demasiado relevantes para la clasificación.

Dentro de esta categoría, las investigaciones han llegado a producir prototipos funcionales bastante precisos a pesar de que la evaluación de su desempeño y de su habilidad para cumplir los objetivos establecidos aún se encuentra en experimentación. Actualmente, existen a nivel comercial algunos productos catalogados como sistemas manejadores de base de datos distribuídas y homogéneas. Aunque ninguno cumple cabalmente con todos los requerimientos y funciones establecidas, puede afirmarse que representan un gran apoyo para la construcción de bases de datos distribuídas en ambientes organizacionales. Quedan excluidos de este grupo, algunos productos comerciales centralizados a los cuales se les extiende con módulos especiales para que puedan funcionar sobre redes, especialmente de área local. La exclusión se debe a que no proporcionan las principales funciones de transparencia de la distribución y localizacion de datos, características principales de los SMBDD. Referencias sobre estos sistemas se encuentran en los siguientes autores [Ceri 84], [Escu 85]. Los sistemas homogéneos se subdividen en centralizados y altamente acoplados[2]:

2) Los sistemas heterogéneos están compuestos por SMBD locales que manejan modelos implantables diferentes. El SMBDD integra las diferentes visiones individuales en una sola por medio del esquema de distribución. No obstante, este objetivo de integracion es muy difícil de lograr y le agrega gran complejidad de construcción al sistema manejador, especialmente para el logro de una interfaz común entre usuarios. El nivel de heterogeneidad de estos sistemas, varía según se determinan grados de alejamiento[3] entre los nodos integrantes del sistema [Ceri 84], [Date 85]. A nivel de investigación, los sistemas heterogéneos aún presentan problemas para cumplir con todos

---

[2] No debe confundirse con el concepto de sistemas altamente acoplados, utilizado en sistemas distribuidos. En dicha área, el concepto se utiliza para caracterizar los sistemas de procesamiento en paralelo (con varios procesadores) que comparten la misma unidad de memoria principal. En este caso se utiliza para definir la integración resultante entre SMBD similares.

[3] Relacionados con diversidad de SMBD, modelos implantables de datos, equipos, sistemas de apoyo, protocolos, etc. A medida que hay mas diversidad hay mas alejamiento entre los componentes del sistema distribuido.

los objetivos planteados. Sin embargo, ya existen modelos experimentales que se acercan mucho a lo esperado en el manejo óptimo de dicha heterogeneidad. Como se mencionó, el problema central está en la complejidad de implantación de sus componentes individuales y en la dificultad de encontrar modelos lo suficientemente generales como para soportar cualquier tipo de SMBD local sin degradar el desempeño global del sistema distribuido. Entre los modelos típicos heterogéneos se encuentran, el Multibase, el Sirius-Delta, DDTS, Mermaid y algunos productos comerciales que apoyan la construcción de BDD heterogéneas, tales como Linda y Cics/Isc. Las referencias bibliográficas [Ceri 84], [Date 85], [Virg 85], [Glig 85], [Temp 85], [Wolk 89] describen detalladamente ambos tipos de sistemas.

3- MODELO GENERAL DE RENDIMIENTO PARA SMBDD

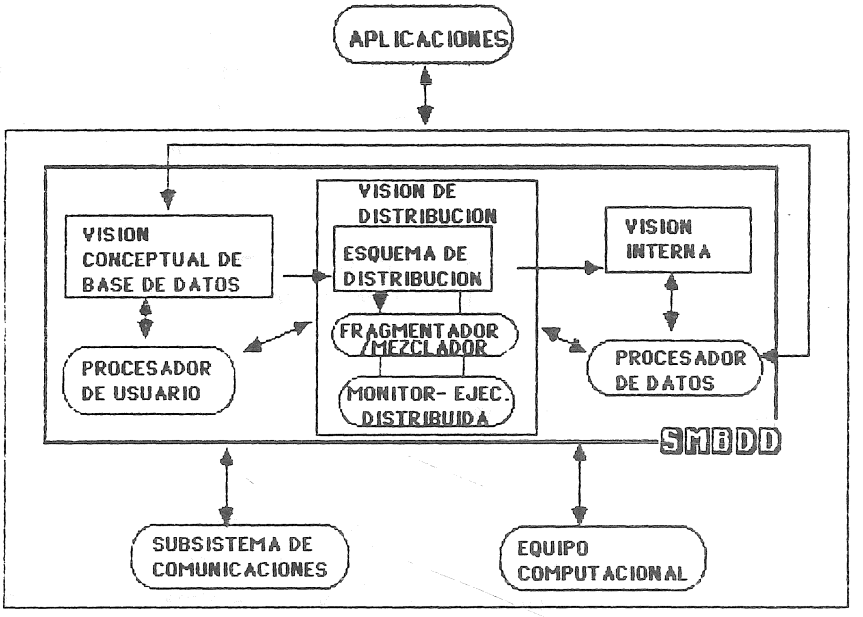
Con el fin de estructurar mejor las ideas seguidas en el estudio, se establece un modelo general de rendimiento que permite mostrar, en forma clara y precisa, los parámetros, relaciones y grado de contribución de cada uno de ellos en el comportamiento (tiempo de respuesta) total del sistema.

Para la construcción de dicho modelo se toma como base la arquitectura mencionada anteriormente y el modelo de rendimiento para bases de datos centralizadas establecido por Ignacio Casas y Kenneth Sevcik [Casa 86]. En la figura anexa, se muestran gráficamente cada uno de sus componentes y las vías de comunicación e interacción entre ellos.

La característica principal del modelo radica en la agrupación de los componentes del SMBDD atendiendo a las funciones lógicas de operación de los componentes que conforman la arquitectura. También se toman en cuenta los factores del medio ambiente (externos) que influyen en el tiempo total de respuesta del sistema. A continuación se describen cada una de las secciones que constituyen el modelo:

3.1-Aplicaciones

Define los parámetros de utilización generados por el ambiente en el cual presta apoyo el sistema distribuído. Se modela en términos de factores o características de los programas que accesan la base de datos distribuída.



MODELO DE RENDIMIENTO

FIGURA # 2.2

Para efectos de rendimiento del sistema, en este grupo se colocan elementos tanto cuantitativos como cualitativos; entre otros están el grado de utilización, tipos de datos manejados, volúmen de datos implicados en cada solicitud y frecuencia de uso de la aplicación. Estos factores son determinados, directamente por el usuario y su ambiente. La influencia de este grupo dentro del comportamiento del sistema, se refleja a través de la traducción de dichas características al modelo conceptual global de datos y a las operaciones realizadas sobre el conjunto de datos; pero muy especialmente, se refleja en el diseño de la base de datos distribuida, ya que determina la forma óptima de distribución, el grado de fragmentación y de replicación de datos; así como también, interviene en la selección de la estrategia distribuida más adecuada en un momento dado.

3.2- **Visión Conceptual** -esquemas externo y conceptual de la base de datos-.

En este grupo del modelo de rendimiento, se ubican los elementos de la base de datos que estan relacionados con la representación conceptual y externa del modelo de datos. Dicha representación dependerá del modelo lógico[4] seleccionado y de características particulares de los manejadores de base de datos locales. Los usuarios de la base de datos pueden tener diferentes representaciones logicas de los datos, estas son llamadas visiones externas. Por lo tanto, el modelo conceptual debe contemplar la integración de estas visiones individuales y debe proporcionar medios de conversión y manipulación de datos cuando las visiones concentradas difieren mucho unas de otras.

3.3- Visión de Distribución

Relacionada con el esquema de distribución y el conjunto de factores requeridos para la determinación de la estrategia de ejecución distribuída. Por lo tanto, se incluyen en ella el esquema de localización y distribución de datos, el cual es originado por el diseño de la BDD [Ceri 84], [Ceri 87]. Dicho esquema contiene las direcciones de los nodos que almacenan los conjuntos de datos requeridos para satisfacer las diferentes solicitudes de acceso del usuario. Se incluye la descripción de tipo y grado de fragmentación y replicación de datos. Este esquema es utilizado por otro elemento incluído dentro de esta sección, el fragmentador, el cual se encarga de hacer efectiva la ejecución distribuída. El monitor de ejecución distribuída es el encargado de monitorear y controlar el procesamiento distribuído. Debido a su relación con los elementos anteriores es incluído como tercer componente de la visión distribuída. Estos elementos son considerados de gran repercusión sobre el comportamiento esperado en el SMBDD.

3.4- Visión Interna

El esquema interno corresponde al diseño físico de los datos almacenados en cada uno de los nodos locales. Este está estrechamente relacionado con la implantación de la base de datos y con los mecanismos de acceso a dichos datos. En un sistema distribuído son particulares del SMBD local, por lo tanto

---

[4] Revisar alguna de las siguientes referencias, en las cuales se describen dichos modelos [Kort 87], [Ever 86].

las estrategias para selección y acceso al conjunto de datos también se incluyen dentro de la visión interna. Dado que corresponde a la representación de los datos en los dispositivos de almacenamiento, su organización efectiva depende de los formatos, agrupamientos, enlaces y codigos de representación que permite un equipo dado.

3.5- Procesador de datos

Se refiere a las características particulares del SMBD local para la realización de las manipulaciones necesarias sobre los comandos físicos y canónicos. Es decir, lo relacionado a programas de conversión de datos. También la validación de consistencia e integridad de su base de datos local y de informar sobre el monitoreo de la ejecución llevada a cabo en su nodo.
Existe una relación directa con el equipo computacional y las facilidades que brinda para el manejo y acceso de datos y archivos a nivel físico. El SMBD debe optimizar el uso del equipo aprovechando sus ventajas para obtener mejor rendimiento de procesamiento.

3.6- Procesador de usuario

Contempla todo lo relacionado con los mecanismos de traducción y formateo de comandos en el nivel de usuario. Su punto clave son los algoritmos de manipulación de consultas. La forma de generar subconsultas a partir de una consulta global tiene fuerte influencia en el rendimiento de todo el sistema. El procesador de usuario (cliente) tiene una estrecha relación con los esquemas conceptual y externo de la base de datos. Y su salida va directamente a la visión distribuída a traves de la cual será encaminada apropiadamente.

3.7- Equipo Computacional

Se refiere al equipo físico de apoyo al sistema. Son los procesadores sobre los cuales corren los programas del sistema distribuído. Por lo tanto, de su desempeño depende el grado de respuesta del SMBDD. Se incluyen en este grupo, procesadores, controladores, dispositivos de entrada/salida, de almacenamiento, junto con sus respectivas características particulares de manipulación. También forman parte los programas de apoyo del sistema, tales como sistemas operativos los cuales son básicos para el funcionamiento

normal del sistema computacional. Establecen limitantes fuertes en las implantaciones de SMBD, las cuales pueden extenderse a los SMBDD.

3.8- Subsistema de comunicaciones

Dentro de esta clasificación se colocan los elementos necesarios, tanto de equipo (hardware) como de programas (software), para el transporte de los datos o resultados de procesamiento entre las diferentes localidades. Y además, todos los elementos encargados de las comunicaciones de cualquier otro tipo de mensajes entre los componentes del sistema distribuído.
Las caracteristicas de implantación y manejo del equipo y protocolos de comunicaciones son excluídas de este grupo. Para los efectos del SMBDD, solo se requieren a nivel de recepción y entrega de datos, por lo tanto sólo se incluyen los detalles referentes a la interfaz de comunicación. Se supone la existencia de un front-end de comunicaciones en cada localidad.

4- DEFINICION DE LAS ECUACIONES BASICAS DE RENDIMIENTO

Los parámetros seleccionados para la evaluacion son aquellos que influyen en la medida del tiempo de respuesta. Por lo tanto, el conjunto de parámetros estan relacionados con el tiempo de procesamiento ocupado por el SMBDD para la realización de cada una de las actividades relacionadas con la manipulacion de la BDD. La suma de todas esas cantidades individuales conforman el tiempo global de respuesta medido para el sistema distribuido. Se utiliza al modelo de rendimiento, definido en la sección anterior, como base para la determinación y agrupamiento de los parámetros de evaluación.

Para derivar las ecuaciones del tiempo de respuesta se toma como base la secuencia de procesamiento que sigue una solicitud de acceso a la base de datos distribuída y el conjunto de parametros derivados para cada una de los componentes del modelo de rendimiento. De ésta manera, el tiempo total de respuesta se fragmenta en:

1-Tiempo de visión conceptual	(TVC)
2-Tiempo de visión distribuída	(TVD)
3-Tiempo de procesamiento local	(TPL)
4-Tiempo de equipo computacional	(TEQ)
5-Tiempo de comunicaciones	(TCOM)

(1)- El (TVC) **tiempo** de **visión conceptual** esta conformado por los factores [Gil89], [Hevn 85], [Lars 85b]:

- tiempo de lectura al esquema conceptual depende del tamaño del esquema **T**, del método de acceso al esquema **A**:

$$t_{\text{lectura}} (T, A) \quad (1)$$

- tiempo de revisión de integridad depende del momento en que se realice la revisión, previo, en paralelo o posterior a la ejecución de la solicitud

$$t_{\text{integridad}} \quad (2)$$

- tiempo de traducción depende del lenguaje manipulador de datos **LMD** y del algoritmo utilizado para la traducción **AL**

$$-t_{\text{traducción}} (AL, LMD) \quad (3)$$

- tiempo de localización de los datos - contención por bloqueo de la base de datos realizada por los mecanismos de control de concurrencia

$$t_{\text{localización}} \quad (4)$$

- tiempo de acceso a esquema externo

$$t_{\text{externo}} (\text{tamaño del esquema, método de acceso}) \quad (5)$$

-tiempo de formateo de resultados depende del lenguaje manipulador de datos

$$t_{\text{formateo}} \quad (6)$$

La ecuación total del tiempo para la visión conceptual es:

$$TVC = t_{\text{lectura}} + t_{\text{integridad}} + t_{\text{traducción}} + t_{\text{localización}} + t_{\text{externo}} + t_{\text{formateo}} \quad (7)$$

(2)- El (**TVD**) **tiempo** de **visión distribuída** está dado por los siguientes factores [Jark 85], [Daya 85], [Chu 82], [Yu 85]:

-tiempo de acceso distribuido, depende del tamaño del esquema de distribución **TD**, de la cantidad de datos implicados **C**, métodos de acceso **A**

$$t_{\text{acceso distribuído}} (TD, C, A) \quad (8)$$

-tiempo de estrategia distribuída, depende de los parámetros definidos **C**, de la distancia entre nodos **D**, grado de replicación **R**, número de nodos que contienen datos requeridos **N**, secuencia de operaciones **S** y materialización de las mismas **M**:

$t_{estrategia\ distribuida}$ (C, N, D, R, S, M)[5] (9)

- tiempo de mezcla de resultados, depende del número de operaciones distribuídas **OD** y, orden de ejecución **O**. Se determinan dos tiempos importantes:

*tiempo de mezcla: t_{mezcla} (OD, O) (10)

*tiempo de mezcla-real : implica el tiempo que debe esperar para completar la actividad de mezcla de resultados, desde que comienzan a llegar los resultados parciales de los diferentes nodos hasta que obtiene el resultado total en el formato deseado

$$t_{mezcla\text{-}real} = t_{local\text{-}último} - t_{local\text{-}primero} + t_{mezcla} \quad (11)$$

Así el tiempo de visión distribuída queda definido como:

$$\mathbf{TVD} = t_{acceso\text{-}distribuído} + t_{estrategia} + t_{mezcla\text{-}real} \quad (12)$$

(3)- El (**TPL**) **tiempo de procesamiento local**, esta definido por los siguientes factores [Lars 85c], [Jark 85], [Onue 88], [Marc 85]:

-tiempo de acceso a esquema externo

$$t_{acceso\text{-}externo} = t_{externo} + t_{integridad} \quad (13)$$

-tiempo conversión de comandos esta es una definición general, luego a nivel de cada SMBDD, se determinan las peculiaridades de cada uno de ellos, depende de los siguientes tiempos

*tiempo de acceso a esquemas internos

$$t_{interno}\ (TI, A) \quad (14)$$

*tiempo de manipulación sobre los datos, el cual depende del lenguaje manipulador de datos **LMD** y de la cantidad de datos implicados **C** :

$$t_{manipulación}\ (LMD, C) \quad (15)$$

$$t_{conversión\ de\ comandos} = t_{interno} + t_{manipulación} \quad (16)$$

-tiempo de acceso a la base de datos física, el cual depende del tamaño real de los archivos internos **TBD**, de la cantidad de

---

[5] En los estudios realizados por [Mukk 88] , [Ceri 84] se definen fórmulas muy específicas para manipular las variables mencionadas. Son válidas únicamente para casos especiales, donde se ha restringido deliberadamente el contexto ; para obtener estimaciones mucho más precisas.

datos implicados **C** y de las estructuras de apoyo para la implantación y acceso más rápido a la bd física **EA**.

$$t_{\text{acceso físico}} (TBD, C, EA) \quad (17)$$

- tiempo de formateo a comandos canónicos, depende directamente de las expresiones canónicas **EC** utilizadas y del lenguaje manipulador de datos **LMD** que se tenga

$$t_{\text{formateo}} (LMD, EC) \quad (18)$$

$$TPL = t_{\text{acceso-externo}} + t_{\text{conversión}} + t_{\text{acceso físico}} + t_{\text{formateo}} \quad (19)$$

(4)- El (**TEQ**) **tiempo** de **equipo computacional** esta dado por los siguientes elementos [Kein 86], [Vija 85], [Papa 86]:

- tiempo de ejecución de transacciones, dado por el ciclo de instrucción del procesador central de la máquina que apoya la ejecución distribuida

$$t_{\text{cpu}} \quad (20)$$

- tiempo de servicio o acceso al disco para accesar la base de datos, depende de los tiempos de rotación, latencia y posicionamiento de la cabeza lectora, así como del número de bloques que lee por cada acceso y, finalmente por el número de veces que debe repetir la actividad para completar una transacción de lectura o escritura sobre la base de datos (**n**). Para los efectos de este estudio se considera como un factor integrado de todos estos elementos:

$$t_{\text{servicio total}} = n * t_{\text{servicio}} \quad (21)$$

- tiempo de retraso por bloqueo del medio, causado por mecanismos de control de concurrencia y de atomicidad de las transacciones

$$t_{\text{bloqueo-disco}} \quad (22)$$

$$TEQ = t_{\text{cpu}} + t_{\text{servicio total}} + t_{\text{bloqueo-disco}} \quad (23)$$

este tiempo afecta a cada uno de los elementos del modelo que realiza algún tipo de procesamiento.

(5)- El (**TCOM**) **tiempo** de **comunicaciones** esta definido por elementos tales como [Hopp 89], [Hutc 88]:

- tiempo de transmisión, los factores más relevantes que lo afectan son: el volúmen de datos transmitidos **V**, de la distancia **D** que debe recorrerse y la capacidad del canal de comunicaciones **CC**:

$$t_{transmisión}(V, CC, D) \quad (24)$$

- tiempo de acceso al canal de comunicaciones, depende de los protocolos **P** que se establecen y de la topología de la red **TR** que apoya la BDD

$$t_{acceso\ canal}(P, TR) \quad (25)$$

- tiempo de retraso introducido por congestión en la red debido a sobrecargas en el medio de comunicación

$$t_{retraso} \quad (26)$$

- tiempo de servicio de comunicaciones, utilización de front-end de comunicaciones el cual se encarga de todos los factores de interfaz que son ajenos al SMBDD, pero que son necesarios para lograr el procesamiento distribuído. Depende directamente de la velocidad del **CPU** y de los protocolos **P** utilizados para lograr la interfaz:

$$t_{ser.comunicación}(CPU, P) \quad (27)$$

$$\mathbf{TCOM} = t_{transmisión} + t_{acceso\ canal} + t_{retraso} + t_{ser.comunicación} \quad (28)$$

Tomando en cuenta las ecuaciones individuales definidas, se establece la ecuación final para el tiempo de respuesta del SMBDD. Para definirlo se ordena en base al ciclo de procesamiento lógico seguido por las solicitudes. El **tiempo total TT** es:

$$TT = [(\text{Tiempo procesador-cliente}) + TCOM + (\text{Tiempo proceso local})]*TEQ \quad (29)$$

El tiempo en procesador cliente, está constituído por los tiempos necesarios para ejecutar las actividades en el nodo procesador de la solicitud de acceso a la base de datos distribuída.

El tiempo de proceso local también está conformado por la suma de los tiempos de las actividades requeridas, para la producción de resultados parciales a la solicitud de usuario.

5- EVALUACION DE RENDIMIENTO PARA CADA UNO DE LOS DIFERENTES TIPOS DE SMBDD [Barr 90]

El modelo anteriormente expuesto, fue ajustado a las caracteristicas de procesamiento de cada uno de los modelos de SMBDD. Se hicieron las diferenciaciones sobre cada una de las ecuaciones involucradas en dicho procesamiento. Estos modelos particulares fueron sometidos a la influencia

de las aplicaciones relacionadas con Sistemas de Informacion (administrativos y de apoyo a la toma de decisiones), Sistemas Integrados (manufactura, control de procesos y de produccion) y Sistemas Transaccionales.

De la evaluacion se destacan los siguientes puntos:

1- El tiempo de estrategia distribuída, ha sido denotado como el más relevante para la obtención de un tiempo mínimo de respuesta, independientemente del tipo de SMBDD. Sin embargo, su contribución al tiempo total, no está definida como el tiempo mínimo que toma efectuar una decisión estratégica de operación distribuída; sino en el tiempo que dicha decisión ahorra en términos de mayor efectividad en el uso de los recursos que componen el SMBDD. Así, se entiende que si un TVD, es ligeramente alto al compararlo con otros tiempos del modelo, éste se justifica completamente; dado que al producir un efecto reductor en otros tiempos más impactantes, tales como procesamiento local y comunicaciones, el tiempo global de respuesta será menor.

2- El caso contrario, implica que si se dedica menos tiempo a la definición de estrategia, puede ahorrarse tiempo de TVD, pero la falta de optimización requerirá mayor utilización de otros recursos del SMBDD y si estos recursos son limitados, el rendimiento total del sistema irá degradando su valor hasta caer en un estado de completa ineficiencia desde todo punto de vista.

3- Debe quedar muy claro que lo esencial en un SMBDD es la cooperación entre componentes. Debe lograrse el equilibrio entre ellos, para garantizar la supervivencia[6] del sistema completo, tal como ocurre con cualquier otro tipo de sistema. El rango aceptable de valores para este tiempo total es establecido por las características operativas demandadas por las aplicaciones.

4- Es muy cierto que los factores externos tales como equipo y sistema de comunicaciones son importantes para el logro de un buen rendimiento de SMBDD; pero se han destacado factores que limitan esta

---

[6] Dentro de este contexto supervivencia significa utilidad máxima. Proporcionar los beneficios de utilización y operatividad adecuados y establecidos por el ambiente en el cual se encuentra y al cual presta apoyo.

aceleración del tiempo de respuesta, tales como la estrategia distribuída y optimización de consultas. Los cuales causan una subutilizacion de recursos, deficiencia en la asignacion de tareas que al manejarse en forma aislada sin importar los demas componentes del sistema originan un incremento notable en el tiempo de respuesta del sistema.

Los resultados obtenidos pueden ser resumidos en las siguientes tablas (la nomenclatura utilizada es la misma definida en el modelo de rendimiento):

SMBDD's	SISTEMAS DE INFORMACION	SISTEMAS INTEGRADOS	APLICACIONES TRANSACCIONA.
Hom.Alt.Acoplado	Adecuado	Adecuado	Adecuado
Hom.Centralizado	No - Adecuado	No - Adecuado	Adecuado *
Heterogéneo	Adecuado *	Adecuado	Adecuado *

TABLA 1: SMBDD .vs. APLICACIONES

(**Adecuado** : presenta las caracteristicas necesarias para que la aplicación funcione apropiadamente.

Adecuado * : puede ser utilizado por la aplicación pero con algunas limitaciones.

No-adecuado: no presenta las características básicas requeridas para soportar la aplicación.)

	SISTEMAS DE INFORMACION	SISTEMAS INTEGRADOS	APLICACIONES TRANSACCION.
TVC	RELEVANTE	IMPORTANTE	NORMAL
TVD	RELEVANTE	IMPORTANTE	NORMAL
TPL	NORMAL	NORMAL	NORMAL
TEQ	NORMAL	RELEVANTE	NORMAL
TCOM	NORMAL	RELEVANTE	IMPORTANTE

TABLA 2 : SMBDD Homogéneo Altamente Acoplado

TVC	R	R	I
TVD	I	I	N
TPL	I	I	N
TEQ	N	I	I
TCOM	N	R	I

TABLA 3: SMBDD Heterogeneo

	Central	Servicio	Central	Servicio	Central	Servicio
TVC	I	-	N	I	N	N
TVD	I	-	N	I	N	I
TPL	N	-	N	-	N	-
TEQ	R	I	R	R	N	N
TCOM	N	I	R	R	I	I

TABLA 4: SMBDD Homogéneo Centralizado

(**Relevante** : afecta fuertemente el tiempo total de respuesta del sistema.

Importante: no debe descuidarse es básico en la determinación del tiempo total.

Normal : su influencia sobre el tiempo de respuesta no es superior a la de los demás elementos.

No es importante: influencia mínima sobre el tiempo de respuesta).

6- CONCLUSIONES Y RECOMENDACIONES

1) A través del análisis efectuado se determinó que los factores principales de rendimiento de un SMBDD están dentro de la sección de visión distribuida perteneciente al modelo de evaluación definido.

2) La optimización del acceso remoto se fundamenta en el diseño conceptual de la BDD, por lo tanto la relación de rendimiento es directamente proporcional a la implantación de un buen diseño de la misma.

3) Del punto anterior se concluye que no puede haber un alto rendimiento en un sistema de BDD, cuyo diseño de datos sea deficiente. Las estrategias de optimización de consultas y de ejecución distribuída solo pueden mejorar hasta cierto grado la respuesta del sistema. Si la distribución de los datos no es la óptima para enfrentar las características de la aplicación que la utiliza, el rendimiento será minoritario o al menos no podrá superar ciertos límites establecidos por otros parámetros del ambiente en el cual se encuentra el sistema de base de datos.

4) La influencia de los medios de comunicación no es tan fuerte en el rendimiento de los SMBDD. Aún contando con un canal de comunicación de gran capacidad, la estrategia puede determinar un tiempo de respuesta bajo si esta hace un uso excesivo de los medios de comunicación.
Mientras no se saturen los canales de comunicación la influencia en el tiempo de respuesta es solamente determinada por el número de veces que estos se utilicen. La limitación en el tiempo de respuesta ocasionada por los medios de comunicación, solo se alcanza cuando los canales no son apropiados o no son capaces de manipular el volúmen y tráfico de datos requerido por las aplicaciones.

5) La diferencias, con respecto a los parámetros de evaluación de rendimiento entre los sistemas homogéneos y los heterogéneos, se concentra en hacer mas enfasis en la optimizacion de consultas. La diferencia real se establece a nivel de implantación o construcción de los mecanismos que garantizan un funcionamiento transparente del sistema ante los usuarios. La complejidad está en la forma de lograrlo, no en la forma de evaluarlo.

6) El estudio presentado a través del modelo de rendimiento permitió identificar un conjunto de elementos individuales que contribuyen a la obtención de un tiempo global de respuesta. Las relaciones entre dichos componentes no ha sido exhaustivamente estudiada y analizada aquí, por lo que constituyen un punto de inicio para estudios posteriores.

7) A traves de modelos de simulacion se validaron algunos de los resultados teoricos senalados en el estudio. Actualmente se esta trabajando en la validacion y profundizacion de los conceptos expuestos en el modelo.

7- BIBLIOGRAFIA

[Barr 90] Modelo para la evaluacion de rendimiento de Sistemas Manejadores de Base de Datos Distribuidas. Tesis de Maestria. ITESM. Monterrey, Mexico 1990.

[Bhar 89] Bhargava Bharat; Mafla Enrique; Riedl Jonh; Sauder Bradley. Implementation and measurements of efficient Communication Facilities for Distributed Database Systems. IEEE Fifth International Conference on Data Engineering 1989.

[Bhid 88] Bhide Anupan.; Stonebraker Michael. A performance comparision of two architectures for fast transaction processing. IEEE Fourth International Conference on Data Engeneering 1988.

[Bodo 88] Bodorik P ; Riordon J. Distributed query processing optimization objetives. IEEE Fourth International Conference on Data Engineering 1988.

[Casa 86] Casas Ignacio; Sevcik Kenneth. Performance modeling of Database Systems: a framework and a case of study. Computer Systems Research Institute, University of Toronto, Canada. Tech. Report. 1986.

[Casa 89] Casas Ignacio; Sevcik Kenneth. A buffer Management Model for Use in Predicting Overall Database System Performance. IEEE Fifth International Conference on Data Engeneering 1989.

[Ceri 84] Ceri Stefano., Pelagatti Giuseppe: Distributed Databases. Principles & systems" Mc Graw Hill 1984.

[Ceri 87] Ceri Stefano; Percini Barbara; Widerhold Gio. Distributed Database Systems 1987.

[Chu 82] Chu W. W.; Hurley P. Optimal query processing for distributed database systems. IEEE Transactions on computers. September 1982.

[Date 85] Date, C J. An introduction to Database systems. Volumen II. Addison Wesley 1985.

[Daya 85] Dayal Umeshwar. Query proceesing in a multidatabase system. Query processing in Database Systems. Editado por Kim Won,.Reiner David,.Batory Don.1985.

[Ever 86] Everest Gordon. Database management objectives, systems functions & administration. McGraw Hill. Computer sciences series. 1986.

[Escu 85] Esculier Christian. The Sirius -Delta architecture: a framework for co-perating database systems. Tutorial : Distributed database management. IEEE Computer Society 1985.

[Garc 88] Garcia Molina, Hector; Kabbott Robert. Reliable distributed database management. Proceedings of IEEE. May 1987.

[Gill 89] Gillenson, Mark. Introducción a las base s de datos. McGraw Hill 1989.

[Hevn 85] Hevner Alan. Methods for data retrieval in distributed systems. Tutorial. Distributed database management. IEEE computer Society 1985.

[Hevn 87] Hevner Alan. R., Yao S. Bing. : Quering distributed database on local area network: Proceedings of the IEEE. May 1987.

[Hopp 89] Hopper; Temple; Williamson. Diseño de redes locales.. Adisson Wesley. Iberoamericana S.A. 1989.

[Hutc 88] Hutchinson David. Local area network architectures. International computer science series. Addison Wesley 1988.

[Jark 85] Jarke Matthias; Jurgen Koch; Schmidt Joachim. Introduction to query processing. Query processing in Database Systems. Editado por Kim Won,.Reiner David,. Batory Don.1985.

[Kein 86] Kleinrock Leonard. Performance models for distributed systems. Teletraffic Analysis and computer performance evaluation. Elseviers Science Publisher B.V. 1986.

[Kort 87] Korth, Henry; Siverschatz, Abraham. Fundamentos de base de datos. McGraw Hill. 1987.

[Lars 85a] Larson James. Introduction. Tutorial. Distributed database management. IEEE computer Society 1985.

[Lars 85b] Larson James. Bringing the gap betwen network and relactional database. Special feature. Tutorial. Distributed database management. IEEE computer Society 1985.

[Lars 85] Larson James; Rahimi Saeed. Transforming database command . Tutorial. Distributed database management. IEEE computer Society 1985.

[Marc 85] March Salvatore; Carlis John. Physical Database Design: Techniques for improved Database Performance. Query processing in Database Systems. Editado por Kim Won, Reiner David, Batory Don.1985..

[Mukk 88] Mukkamala Ravi. Measuring the effect of data distribution and replication models on performance evaluation of distributed database systems. IEEE Fourth International Conference on Data Engeneering 1988.

[Nich 88] Nicholls Bill. The "B" world. BYTE June 88.

[Onue 88] Onuegbe Enmanuel; Rahimi Saeed; Hevner Alan. Local query translation and optimization. IEEE Fourth International Conference on Data Engeneering 1988.

[Papa 86] Papadimitrou Christos. The theory of Database Concurrency Control. Computer Science Press. 1986.

[Robi 88] Robie Jonathan. Fast data access. Byte January 1988.

[Vija 85] Vijay Ahuja. Design and analysis of Computer Communication Networks. International Student Edition. McGraw Hill 1985.

[Virg 85] Virgil Gligor; Luckenbaugh Gary. Interconecting heterogeneos database management systems. Tutorial Distributed database management. IEEE computer Society 1985.

[Youn 88] Young Jeffrey; Warren Carl; Mac user Lab staff. Database comparision. MacUser December 1988.

[Yu 85] Yu Clement. Distributed query proceesing. Query processing in Database Systems. Editado por Kim Won, Reiner David, Batory Don.1985.

Ambiente de Especificación Denotacional

Pedro Borges
Roger Soler

Departamento de Computación y
Tecnología de la Información
Universidad Simón Bolívar
Apartado Postal 89000
Caracas, Venezuela

Resumen

En este trabajo presentamos un ambiente basado en el método denotacional para la definición de lenguajes de programación. El ambiente permite manipular o ejecutar especificaciones denotacionales mediante su concretización en LISP. Presentamos una breve descripción de las especificaciones que el ambiente maneja y la descripción funcional del ambiente.

1. INTRODUCCION

El ambiente propuesto tiene como objetivo manipular especificaciones denotacionales, las cuales están basadas en el Método Denotacional desarrollado por D. Scott y C. Strachey ([Stoy77], [Gordon79]), para especificar la semántica de los lenguajes de programación secuenciales.

Este método consiste en asociar a cada constructor sintáctico del lenguaje de programación un valor u objeto matemático. Esta asociación está dirigida por la morfología del lenguaje y los valores u objetos matemáticos pertenecen a lo que se conoce como Teoría de Dominios ([Scott76], [Soler82]). Para lograr la manipulación de estos objetos, el ambiente crea aproximaciones a los mismos mediante estructuras de un lenguaje funcional (LISP).

La especificación denotacional presenta la descripción abstracta del interpretador de un lenguaje. Varios trabajos se han orientado a la generación de compiladores "eficientes" a partir de dichas especificaciones ([Bodwin82], [Raskovsky82], [Sethi83], [Wand82], [Kelsey89]). En nuestro caso, se ha buscado que la implementación sea lo más próxima posible a la descripción abatracta, principalmente manteniendo la estructura funcional del interpretador abstracto. Esto nos permite, en particular, observar el funcionamiento de las partes del interpretador.

A continuación se presenta una breve descripción de las especificaciones denotacionales que manipula el ambiente, luego la descripción funcional del ambiente y algunos comentarios sobre el mismo.

2. ESPECIFICACION DENOTACIONAL

2.1. Dominios sintácticos.

La especificación de los dominios sintácticos consta de dos partes. La primera es una lista de los nombres de los dominios y sus correspondientes elementos genéricos. La segunda es la descripción de la morfología de dichos elementos genéricos, en forma de "operador(operandos)", de modo de eliminar ambigüedades. Por ejemplo, la morfología de una instrucción del tipo "si exp entonces inst_1 sino inst_2" sería de la forma "si(exp,inst_1,inst_2)".

Esta notación para la morfología de los dominios sintácticos se considera equivalente a la descripción de la forma de los árboles abstractos de los elementos de dichos dominios, donde el operador se considera el nodo raíz y los operandos los nodos hijos.

Los dominios de identificadores y cifras están predefinidos con ciertos operadores. En particular, la igualdad sintáctica de identificadores.

2.2. Dominios semánticos.

Esta especificación es una lista de las variables genéricas, los nombres y las estructuras de cada uno de los dominios

semánticos que se definen. Los dominios de los números enteros y los valores de verdad (cierto, falso) se consideran dominios primitivos con nombre y estructura predefinidas, por lo que no es necesario especificarlas. Las operaciones usuales sobre estos dominios también están predefinidas: aritméticas (suma, resta, multiplicación, ...), relacionales (igualdad, menor, mayor, menor o igual, ...) y lógicas.

En general la estructura de un dominio se describe a partir de otros dominios mediante el uso de los constructores producto cartesiano(X), unión disjunta(+), espacio de funciones(->) y secuenciación como generalización del producto cartesiano(*); para los cuales se dispone de un conjunto de operaciones tales como: selección de una componente en un producto cartesiano (↓), proyección en un componente de una unión (|), aplicación de funciones, cabeza y cola de una secuencia, etc.

Típicamente, el significado o interpretación de los programas en el lenguaje que se define es una función que transforma elementos de un dominio de entrada en elementos de un dominio de salida o respuesta. Estos dominios se consideran "elementos distinguidos" o "dominios de interés".

2.3. Funciones de Interpretación.

La especificación de una funcion de interpretación consiste de su funcionalidad y su definición utilizando el lambda cálculo con las operaciones mencionadas. Cabe mencionar que los elementos sintácticos a los cuales se aplican las funciones de interpretación se denotan por sus árboles abstractos tal y como fueron especificados en la sintaxis abstracta.

En la Figura 1. se presenta la especificación denotacional de un lenguaje muy sencillo, sin incluir la función de interpretación para las expresiones booleanas. Las funciones subrayadas son funciones predefinidas en el ambiente.

3. DESCRIPCION FUNCIONAL DEL AMBIENTE

3.1. Dominio de las especificaciones.

Las especificaciones denotacionales pertenecen a un dominio **ED** de la forma:

$$\mathbf{ED} = \mathbf{DSIN} \times \mathbf{DSEM} \times \mathbf{FINT}$$

donde **DSIN** son los conjuntos de dominios sintácticos, **DSEM** son los conjuntos de dominios semánticos y **FINT** los conjuntos de funciones de interpretación que pueden ser descritos por una especificación.

```
DOMINIOS SINTACTICOS
    prog  ∈  PROGRAMAS
    ide   ∈  IDENT
    inst  ∈  INSTRUCCIONES
    exp   ∈  EXPRESIONES
    ebool ∈  EXPBOOLEANAS
    n     ∈  CIFRAS
MORFOLOGIA
    prog ::=  inst
    inst ::=  "si" ( ebool, inst_1, inst_2 )
         ::=  ";" ( inst_1, inst_2 )
         ::= "mientras" ( ebool, inst )
         ::= "escribir"(ide)
         ::= "sea" ( "=" ( ide, exp ) , inst )
    exp  ::=  ide
         ::=  "+" ( exp_1 , exp_2 )
         ::=  n
DOMINIOS SEMANTICOS
           BOOL
           RES = NATURALES *
    s  ∈  S = [IDENT -> V] x NATURALES *
    v  ∈  V = NATURALES
           C = S -> S
FUNCIONES DE INTERPRETACION
    P : PROGRAMAS -> RES
    I : INSTRUCCIONES -> S -> S
    E : EXPRESIONES -> S -> V
    EB: EXPBOOLEANAS -> S -> BOOL
DEFINICIONES
P[[ inst ]] = ( I[[ inst ]] < LAMBDA id.0 , nil > ) ↓ 2
I[[ "si"(ebool,inst_1,inst_2) ]] =
    LAMBDA s. EB[[ebool]]s -> I[[inst_1]]s ,I[[inst_2]]s
I[[";"(inst_1,inst_2)]] =
          LAMBDA s. I[[inst_2]] ( I[[inst_1]] s )
I[["mientras"(ebool,inst)]] =
         LAMBDA s. EB[[ebool]]s ->
              I[["mientras"(ebool,inst)]] (I[[inst]] s),
              s
I[["escribir"(ide)]] = LAMBDA s.
       < s ↓1, postfijar (I[[ide]] s) s ↓ 2 >
I[["sea"("="(ide,exp),inst)]] =
         LAMBDA s. I[[inst]] < s↓ 1[v/ide], s ↓2 >
                   donde  v = E[[exp]] s
E[[ide]] = LAMBDA s. (s ↓1) ide
E[["+"(exp_1,exp_2)]] = LAMBDA s.
         suma (E[[exp_1]] s) (E[[exp_2]] s )
E[[n]] = LAMBDA s. nat n
```

Fig 1. Ejemplo de Especificación Denotacional

Una especificación particular se denota **ed**, y sus componentes **dsin**, **dsem** y **fint** respectivamente. Los dominios de entrada y respuesta ("elementos distinguidos") de **dsem** se denotan **ENT** y **RES** respectivamente, esto es:

$$ed \in ED; \quad ed = < dsin, dsem, fint >$$
$$dsin \in DSIN$$
$$dsem \in DSEM$$
$$fint \in FINT$$
$$ENT \in dsem$$
$$RES \in dsem$$

3.2. Chequeo estático de tipos.

El ambiente chequea la correctitud (consistencia) de tipos de las definiciones de las funciones de interpretación y las funciones auxiliares en **fint** según la funcionalidad de las mismas, los dominios sintácticos en **dsin** y los dominios semánticos en **dsem**. En caso de detectar algún error, el ambiente emite un mensaje descriptivo del mismo.

Para simplificar supondremos que el resultado de la verificación es "cierto" si no se detectaron errores o "falso" si se detectaron inconsistencias de tipos. Por lo tanto, si VT es la función verificadora de tipos, se tiene:

```
VT : ED -> { "cierto", "falso" }
VT ed =  si ed contiene inconsistencia de tipos
                entonces "cierto"
         si no
                entonces "falso"
```

3.3. Concretización de los dominios semánticos.

Para manipular los dominios semánticos en **dsem**, éstos se concretizan o representan como estructuras en LISP. Si denotamos **ELISP** al dominio de las estructuras en LISP, y P(**ELISP**) al conjunto de las partes del mismo, podemos definir la función concretizadora de dominios semánticos

CDSEM : **DSEM** -> P(**ELISP**)

De modo que CDSEM aplicada a un conjunto de dominios semánticos genera el conjunto de las estructuras en LISP que representan a cada uno de los dominios semánticos en el conjunto al cual se aplica. En particular, CDSEM **dsem** es el conjunto de estructuras en LISP que representan los dominios semánticos en **dsem**.

3.4. Concretización de las Funciones de Interpretación.

La concretización de las funciones de interpretación produce un programa (en LISP) que engloba la semántica del lenguaje que se define en la especificación. Dado que en la definición de estas funciones intervienen los dominios semánticos, para su concretización es necesario conocer las concretizaciones de dichos dominios. Por lo tanto, si denotamos **PLISP** al dominio de los programas en LISP, y CFINT a la función concretizadora, tenemos:

CFINT : **FINT** -> P(**ELISP**) -> **PLISP**

es decir que CFINT, dado un conjunto de funciones auxiliares y de interpretación (en **FINT**) y las concretizaciones de los dominios semánticos (en **ELISP**) produce un programa en LISP que concretiza la semántica del lenguaje que se define en la especificación. En particular

CFINT **fint** (CDSEM **dsem**)

es el programa en LISP que representa la semántica del lenguaje que se define en **ed** (**ed** = < **dsin**,**dsem**,**fint** >).

Hacemos notar que LISP tiene la particularidad de que los programas no son maá que estructuras manipulables por el lenguaje ([Allen78]). En este sentido, el dominio **PLISP** es equivalente al dominio **ELISP**, sin embargo, los denotamos en

forma distinta para obtener mayor claridad en la descripción.

3.5. Acción global del Ambiente.

Podemos definir una función de concretización global en base a las funciones descritas anteriormente, como la función CONC, que realiza el chequeo estático de tipos, y de no encontrar errores, produce la concretización de la semántica contenida en la especificación que se maneja. En caso de encontrar inconsistencias de tipos, CONC regresa un valor especial que denotaremos "error". Tenemos entonces:

```
CONC : ED -> PLISP + { "error" }
CONC ed = si VT ed
                entonces CFINT fint ( CDSEM dsem )
          si no
                entonces "error"
donde  ed = < dsin,dsem,fint >
```

3.6. Ejecución de la semántica.

El programa (en LISP) generado por CONC para una especificación **ed** constituye un interpretador para el lenguaje descrito en **ed**, que puede ser ejecutado por una máquina LISP, la cual puede ser una máquina real o,

típicamente, simulada por software. En todo caso, la podemos representar por la función EVAL de LISP. De modo que:

EVAL (CFINT **fint** (CDSEM **dsem**))

es un interpretador para el lenguaje descrito en **ed**.

Este interpretador recibe la concretización (en LISP) de un programa en el lenguaje definido por **ed** y de un elemento del dominio **ENT** (de **ed**) y produce una concretización de un elemento del dominio **RES** (de **ed**). Este elemento es el generado por la aplicación del programa recibido sobre el elemento de **ENT**. Por lo que tenemos:

EVAL (CFINT **fint** (CDSEM **dsem**)) : **ELISP -> ELISP -> ELISP**

donde el primer elemento en **ELISP** es la concretización de un programa en el lenguaje que se define (en **ed**), el segundo es la concretización de un elemento del dominio de entrada **ENT** y el tercero la concretización de un elemento del dominio de salida **RES**. Para generar las concretizaciones que recibe el interpretador como argumentos, se definen nuevas funciones que se describen a continuación.

Dada una especificación **ed** = < **dsin**,**dsem**,**fint** >, denotaremos **L(dsin)** al lenguaje descrito por **dsin**, es decir, al conjunto de programas (árboles abstractos) del lenguaje definido por

ed, y **prog** a un programa en particular (es decir, **prog** $\in$ L(**dsin**)). Si CPROG es la función concretizadora de programas en L(**dsin**) a LISP, tenemos:

CPROG : L(**dsin**) -> **ELISP**
CPROG **prog** = estructura en LISP que representa a **prog**
donde **prog** $\in$ L(**dsin**)

En cuanto a la concretización de los elementos del dominio **ENT**, si **ent** es un elemento particular (**ent** $\in$ **ENT**), definimos la función concretizadora CENT como:

CENT : **ENT** -> **ELISP**
CENT **ent** = estructura en LISP que representa a **ent**

Utilizando las funciones descritas anteriormente, la aplicación del interpretador a CPROG **prog** y a CENT **ent**, obtenemos la concretización del elemento del dominio de salida **SAL** (es decir, una estructura en LISP que lo representa) generado por la aplicación de **prog** sobre **ent**, según la semántica descrita en **ed**, por lo que:

EVAL(CFINT **fint** (CDSEM **dsem**))(CPROG **prog**)(CENT **ent**) $\in$ **ELISP**

es la concretización del elemento de **SAL** descrito anteriormente.

Si CONC ed es distinto de "error" (y por lo tanto CONC ed ∈ PLISP), la expresión del párrafo anterior es equivalente a:

EVAL (CONC ed) (CPROG prog) (CENT ent) ∈ ELISP

4. COMENTARIOS

Como se mencionó en la sección 2.1, las especificaciones manejadas por el ambiente no describen la sintaxis de los programas "reales" de los lenguajes que definen, sino la morfología de los árboles abstractos de dichos programas. Es sobre estos árboles que el ambiente trabaja. Pensamos incorporar la capacidad de trabajar directamente sobre los programas, para lo que será necesario desarrollar la función de transformación de los programas en sus árboles (lo que se conoce como "parsing"). Esta transformación, sin embargo, es un problema bastante estudiado y para la cual existen diversas técnicas eficientes ([Aho77]).

En cuanto a las funciones de concretización descritas en la sección 3, hemos desarrollado versiones que trabajan sobre especificaciones sin dominios reflexivos ([Naranjo89]), trabajándose actualmente en la incorporación de dichos dominios.

Otro aspecto que se encuentra en estudio es el establecimiento de un mayor número de dominios predefinidos con sus operaciones respectivas como parte del ambiente.

5. CONCLUSIONES

La descripción funcional del ambiente ha permitido identificar claramente las distintas componentes del proceso de implementación de una especificación denotacional. Pensamos que, aún cuando el ambiente no está totalmente desarrollado, será de utilidad para crear especificaciones denotacionales y obtener prototipos de las mismas.

6. REFERENCIAS

[Aho77] AHO, A.V., Y ULLMAN, J.D. Principles of Compiler Design. Addison-Wesley, Reading, Mass, 1977.

[Allen78] ALLEN, J. Anatomy of LISP. McGraw-Hill, 1978.

[Bodwin82] BODWIN, J., BRADLEY, L., KANDA, K., LITLE, D., Y PLEBAN, U. Experience with an experimental compiler generator based on denotational semantics. En Proc. SIGPLAN '82 Symp. Compiler Construction, SIGPLAN Notices 17(6), pp. 216-229 (Junio 82).

[Gordon79] GORDON, M.J.C. The Denotational Description of Programming Languages. Springer Verlag, New York, 1979.

[Kelsey89] KELSEY, R. Y HUDAK, P. Realistic Compilation by Program Transformation. Proc. ACM Symposium on principles of Programming Languages. Enero 1989, pp. 281-292.

[Naranjo79] NARANJO, F. Verificacion de tipos y generación de código a partir de una especificación denotacional. Proyecto de Grado. Tutor: Roger Soler. Universidad Simón Bolívar. Septiembre 1989.

[Raskovsky 82] RASKOVSKY, M. Denotational semantics as a specification of code generators. En Proc. SIGPLAN '82 Symp. Compiler Construction, SIGPLAN Notices 17(6), pp. 230-244 (Junio 82).

[Scott76] SCOTT, D. Data Types as Lattices. SIAM Journal of Computing 5, pp. 522-587, 1976.

[Sethi83] SETHI, R. Control Flow Aspects of Semantics-Directed Compiling. ACM Transactions on

Programming Languages and Systems 5(4), Octubre 1983, pp. 554-595.

[Soler82] SOLER, R. Fundamentos de un Meta-Lenguaje para describir Semántica. Dpto. de Computación, Universidad Central de Venezuela, 1982.

[Stoy77] STOY, J. Denotational Semantics. MIT Press, Cambridge, Mass. 1977.

[Wand82] WAND, M. Deriving Target Code as a representation of continuation semantics. ACM Transactions on Programming Languages and Systems 4(3), Julio 1982, pp. 496-517.

Sobre especificaciones y modelos

C. Zoltan de Torres
Departamento de Computación y Tecnologí de la Información
Universidad Simón Bolívar
Apartado 89000
Caracas 1080
Venezuela

Marzo 1991

Resumen

Se analiza usando un ejemplo, que la calidad de las especificaciones depende fuertemente del modelo utilizado para describirlas. En particular cada modelo de solución puede aportar propiedades a la especificación, que dificilmente fueran deducibles de la especificación informal. Se presentan cuatro soluciones de un mismo problema, cada una de ellas con características muy distintas desde el punto de vista computacional. La comparación de estas soluciones es difícil sin tener en cuenta la arquitectura donde corra la aplicación, sin embargo cada una de ellas aporta a la especificación información del problema que contribuye a la reusabilidad de las implementaciones desarrolladas, y al test de implementaciones futuras.

Abstract

We show, by means of an example, that the quality of specifications depends strongly on the model used to describe them. In particular, each such model can suggest properties for the formal specification which would be hard to to deduce from an informal description. We present four solutions to the same problem, each with very different characteristics from the computational point of view. The relative efficiency of these solutions is strongly dependent on the underlying

system architecture, nevertheless they provide information that can be useful for software reuse and for testing future developements.

1 Metodología de las especificaciones

El proceso usual de desarrollo de software, comienza con una especificación informal del problema a resolver. La continuación natural de este proceso, para los defensores de las especificaciones formales, es la formalización de la descripción informal.

Existe una variedad muy grande de lenguajes de especificación, y dentro de cada familia existen miembros que se diferencian solamente en la semántica elegida. Algunos lenguajes algebráicos seleccionan ciertos modelos particulares: álgebra inicial, final, finitamente generada. Veremos, usando un mismo ejemplo, que ciertas soluciones computacionalmente interesante se dejan fuera de los modelos de la especificación. Por el otro lado, las soluciones encontradas nos proveen de una mejor caracterización del problema y permite tener una visión de familias de soluciones posibles basadas en propiedades de modelos particulares.

Este trabajo parte de una especificación informal del problema a resolver, elaborando una solución computacional del mismo.

Esta solución nos permite asegurar la existencia de al menos una solución computacionalmente aceptable para el problema. Asimismo la solución nos sirve de punto de partida para tipificar que es lo escencial y que es prescindible en las soluciones del problema dado.

2 El problema

El Problema: especificación informal [Extraído de "Concrete Mathematics"] [KNU2].

El problema de nuestro interés es un problema muy antiguo y conocido por el nombre de un historiador del siglo I: Flavious Josephus.

Durante la guerra Judeo-Romana, Josephus se encuentra formando parte de un grupo de 41 rebeldes. Atrapados en una cueva, por los Romanos, prefiriendo el suicidio a la rendición, los rebeldes deciden formar un círculo y 'suicidar' al tercero en la cuenta hasta que todos los soldados fueran eliminados.

Josephus, contrario al suicidio, hace rápidamente las cuentas de manera de colocar a su amigo y a él mismo en posiciones apropiadas del círculo, de manera que resultasen los últimos elegidos en este proceso de suicidio. Gracias a su habilidad matemática y a su velocidad de cálculo es que Josephus pudo contar la historia.

Nosotros simplificaremos el problema, buscando un mecanísmo rápido de cálculo. Cambiaremos la forma de designar al próximo suicida siendo 2 y no 3 como en el problema original. Asimismo, consideraremos que puede quedar un solo sobreviviente en el círculo después de la operación de suicidio.

El Problema: Especificación formal

Nuestro problema se ha reducido a la especificación de una función J_2. Usaremos para la presentación la sintaxis de PLUSS [GAU].

Se define un nuevo tipo llamado JOSEPHUS, que utiliza como tipos primitivos los tipos EJERCITO (que corresponde a la organización de un conjunto de soldados con un operador total sucesor) y booleano.

spec :	JOSEPHUS		
use:	ejército,bool		
sort:	sobrevivientes		
generated by:			
operations:			
J_2 :	ejército	$\rightarrow$	sobrevivientes
vacío:		$\rightarrow$	ejército
member:	sobrevivientes ejército	$\rightarrow$	Bool
axioms:			
	soldados $\neq$ vacío $\Rightarrow$		
	$member(J_2(soldados), soldados)$	$= true$	
where:	*soldados* :	ejército	
end	JOSEPHUS		

Dado nuestro desconocimiento de la función, definimos como un nuevo **sort** el rango de la misma. Veremos que en la medida que nos familiarizamos con la función estaremos en mejores condiciones de caracterizar este conjunto

de valores. Esta caracterización nos permitirá establecer el predicado de identificación de este subconjunto (la función característica.)

Los axiomas, que son usados para describir las propiedades de las funciones, van a ser pocos, debido a nuestro desconocimiento del comportamiento de la función.

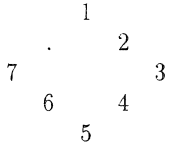
Indicamos en el único axioma de la presentación, que el rango de la función J_2 es un elemento del conjunto inicial de soldados atrapados en la cueva. A continuación veremos diferentes soluciones del problema. Durante el proceso de elaboración de las diferentes soluciones, usando modelos particulares, estaremos en condiciones de completar la especificación de la función. Esto nos permite establecer una jerarquía de especificaciones, mediante la noción de agregar axiomas, usualmente llamado enriquecimiento, operación esta que reduce la cantidad de modelos de la especificación.

La especificación raíz va siendo completada con la alimentación de las especificaciones hoja. A continuación damos las diferentes soluciones al problema, indicando en la subsección 'propiedades derivables de la solución', las propiedades generales de la función que se identifican en la elaboración de las soluciones particulares

SOLUCION 1

Dada la especificación formal la función J_2 esta subespecificada respecto al enunciado del problema, ya que según la especificación, cualquier soldado del conjunto de soldados iniciales puede resultar el sobreviviente (rango de J_2). Una forma de interpretar la especificación es que esta describe la elección de un elemento en un conjunto. Debemos agregar en la especificación la estrategia de eliminación decidida por los rebeldes. A fin de visualizar esa estrategia hacemos una inmersión de los soldados en un intervalo de los naturales. Esta inmersión se limita a asociar los soldados del conjunto con los puestos que ocupan en el círculo. A lo largo de esta solución hablaremos de J_2 como una función cuyo dominio debe interpretarse como el número de puestos en el círculo inicial y cuyo rango corresponde al puesto que ocupa el sobreviviente en el círculo inicial.

Dibujando el círculo y comenzando el proceso de eliminación (no teniendo en cuenta el número total de puestos en el círculo).



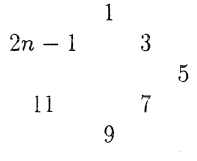
Se elimina cada 2 soldados y si el primero se salva (por el momento), el segundo es eliminado, el tercero se salva y se elimina el cuarto ...

Con este modelo podemos facilmente identificar propiedades del puesto del sobreviviente: es un puesto impar, dado que los puestos pares caen bajo la orden de suicidio en la primera vuelta.

Las órdenes de suicidio de la segunda vuelta dependerán de la paridad del tamaño del círculo inicial. Es de hacer notar que esta caracterización de los soldados suicidados en la primera vuelta no sería posible si hubieramos trabajado con los identificadores de los soldados ya que es un conjunto sin estructura.

Consideraremos ahora las diferentes posibilidades para n el número inicial de soldados.

Para el caso $n = 2m$, y considerando el círculo después de la primera ronda, tendría el siguiente aspecto:



después de haber eliminado al soldado en el puesto $2m$. Este soldado se elimina ya que es par.

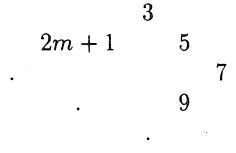
Observando el círculo resultado, podemos considerar que estamos ante la resolución del mismo problema pero con los nombres de los puestos cambiados. Renombrando los puestos y resolviendo el problema, nos resta construir la traducción efectiva de los nuevos puestos a la identificación original.

El nuevo tercer puesto corresponde al quinto puesto del círculo original.

La función inversa de reasignar nombres a los puestos, nos permite establecer la siguiente igualdad:

$$J_2(2m) = 2 * J_2(m) - 1 \tag{1}$$

Para el caso en que el número total de soldados sea impar $n = 2m + 1$: resulta que el soldado con que se inicia la cuenta (soldado 1) es eliminado después del soldado $2m$, resultando que al fin de la primera vuelta mas la eliminación del soldado del puesto 1 queda una configuración que trataremos de hacer corresponder a una configuración inicial:



Si este círculo lo consideramos como una configuración inicial con cambio de nombres. La conversión de puestos resulta en la siguente ecuación:

$$J_2(2m + 1) = 2 * J_2(m) + 1 \tag{2}$$

Las dos relaciones de recurrencia nos permiten establecer un sistema de ecuaciones:

AXIOMA 1	$J_2(2m)$	$=$	$2 * J_2(m) - 1$	$m \geq 1$
AXIOMA 2	$J_2(2m + 1)$	$=$	$2 * J_2(m) + 1$	$m \geq 1$
AXIOMA 3	$J_2(1)$	$=$	1	

Para implementar esta función se deberá considerar las alternativas posibles para el dominio de la función:

1. Los argumentos que se reciben pertenecen a valores posibles.

2. Se debe completar la función para los valores para los cuales no está definida.

Si el tipo del dominio es **cardinal**, debe completarse en 0, y una posible completación es que $J_2(0)$ coincida con la identidad:

$$J_2(0) = 0 \tag{3}$$

Si el tipo de argumento es **entero**, deberemos completar convenientemente para los valores negativos. Hacemos notar que esta completación hace que las álgebras (modelos de la especificación) cambien.

A continuación presentaremos una **implementación** considerando que el dominio de J_2 es cardinal. Supondremos que se dispone de una operación de shift a la derecha. Un posible modelo es 'lista' de booleanos con la interpretación:

$$shiftd(\sum_{0}^{\infty} a_i * 2^i) = \sum_{0}^{\infty} a_{i+1} * 2^i \tag{4}$$

La implementación estará anotada con precondiciones y poscondiciones.

IMPLEMENTACION

PRECONDICION $n \geq 0$
$J_2(n)$
caso n
0 : $J_2 := 0$;
1 : $J_2 := 1$;
sino : **PRECONDICION** $n \geq 2$ por lo que
$shiftd(n) \geq 1$
$A = 2 * J_2(shiftd(n))$
POSCONDICION $A \geq 2, par(A) = true$
si n es *par* entonces $J_2 := A - 1$
sino $J_2 := A + 1$
finsino
POSCONDICION $J_2 \geq 1, impar(J_2) = true$
End J_2

2.1 Propiedades derivables de la solución 1

Hacemos notar que esta implementación depende de un predicado **par**. Este predicado tiene una complejidad que es altamente dependiente del modelo de enteros que se utilice.

Una ley derivable en este momento es:

$$par(J_2(n)) = falso \tag{5}$$

Lo anterior debe inducir a pensar que incluyendo la ecuación anterior en la especificación, siempre se debiera implementar el predicado **par**. Sin embargo la ecuación anterior admite otra presentación:

$$\exists\, i \quad J_2(n) = suc^{2*i+1}(0) \tag{6}$$

La presentación utiliza la función sucesor sobre los naturales. Esta presentación sugiere los modelos de los enteros módulo p para algún p mayor que la talla del ejército. En ese caso la ecuación tiene infinitas soluciones.

SOLUCION 2

Dado que el sistema de ecuaciones recurrentes tiene una expresión cerrada, vamos a hacer el análisis de una implementación usando la expresión cerrada.

IMPLEMENTACION 2

Analizando la implementación 1 vemos que para todo valor de entrada ≥ 1 la operación de $shiftd(n) \geq 1$ y por lo tanto alcanza el valor 1. Es por eso que analizaremos las ecuaciones recurrentes tomandolas de a pares, considerando los casos en que solamente usan los axiomas 1 y 3 y luego los que utilizan los axiomas 2 y 3.

- **Axioma 1 y 3**

  El caso en consideración corresponde a argumentos potencia de 2.

  $J_2(2^r) = 2*J_2(2^{r-1})-1 = 2*(2*J_2(2^{r-2})-1)-1 = 2^2*J_2(2^{r-2})-2-1 = 2^2*(2*J_2(2^{r-3}-1)-2-1 = 2^3*J_2(2^{r-3})-2^2-2^1-2^0$

  y en forma general:

$J_2(2^r) = 2^i * J_2(2^{r-1}) - \Sigma_{j=0}^{i-1} 2^j \qquad i \leq r$

y para el caso $r = i$ resulta

$J_2(2^r) = 2^r * J_2(1) - \Sigma_0^{r-1} 2^j = 2^r - (2^r - 1) = 1$

Los cálculos nos han permitido identificar la propiedad:

$$J_2(2^r) = 1 \tag{7}$$

- **Axioma 2 y 3**

  El formato de los argumentos que solo utilizan los axiomas 2 y 3 es $n = 2^r - 1$

  $J_2(2^r - 1) = J_2(2 * (2^{r-1} - 1) + 1)$

  y usando el axioma 2 resulta

  $J_2(2^r - 1) = 2^i * J_2(2^{r-i} - 1) + \sum_0^{i-1} 2^j \qquad r > i$

  y tomando $i = r - 1$ resulta:

  $J_2(2^r - 1) = 2^r - 1$

Ahora estamos en condiciones de analizar el caso general, resultando

$$J_2(2^r + s) = 2 * s + 1, \; r \geq 0, \; 0 \leq s < 2^r \tag{8}$$

2.2 Propiedades derivables de la solución 2

Vamos a mostrar que este análisis de la implementación de la función de Josephus, nos permite agregar otra ecuación a la especificación. No eran propiedades evidentes obtenibles de la especificación informal de la función.

De la ecuación 8 podemos agregar a la especificación de la función de Josephus:

$$(J_2(n) = 1) = (\exists m \quad n = 2^m) \tag{9}$$

Esta presentación sugiere que una implementación de la función de Josephus pudiera ser utilizada como implementación del predicado que caracteriza a las potencias de 2. Asimismo la ecuación 8 puede ser útil para test de implementaciones que numeren los puestos de 1 a n.

Una pregunta inmediata, es si podemos independizar la función de Josephus de la representación de los enteros. Usaremos la maqueta física de imaginar los soldados en un círculo (esta solución utiliza el modelo sugerido por la especificación informal), e iremos eliminandolos del círculo.

Solución 3

En esta solución retendremos la inmersión de los soldados en el intervalo $[0 \ldots 1]$ y simulamos la operación de suicidio. Por lo tanto nuestro círculo estará formado por soldados vivos y soldados ya suicidados.

La designación de un soldado como el próximo al suicidarse se reduce a encontrar un soldado vivo y suicidar a su vecino (en una dirección determinada) que aún esté vivo.

IMPLEMENTACION

$J_2(n)$
si $n \leq 1$ **entonces** $J_2 = n$

PRECONDICION todos los soldados del grupo estan vivos y comenzará la cuenta con el soldado 1, que está vivo.

	sino	$s := 1; muertos := 0;$
0		**mientras** queden por eliminar
1		$s := s \bmod n + 1$
2		mientras s está muerto $s := s \bmod n + 1$
3		eliminar s; $muertos = muertos + 1$
4		mientras s está muerto $s := s \bmod n + 1$
		finmientras
	$J_2 := s$	

Al ciclo de la línea 0 se ingresa siendo s un soldado vivo, su vecino se obtiene en la línea 1. Este vecino debe ser un soldado vivo (línea 2). En la línea 3 se elimina al vecino vivo y se contabiliza el número de soldados suicidados. La línea 4 recupera la precondición requerida para el ciclo que se inicia en la línea 0. La condición de salida del ciclo se reduce a que el numero de muertos sea uno menos que el total original de soldados.

2.3 Propiedades derivables de la solución 3

La solución presentada esta parametrizada por los enteros módulo el tamaño del grupo inicial de soldados. Este tipo de presentaciones no se admite en un gran número de lenguajes de especificación.

Nuestra especificación admite como modelos álgebras finitas (por ejemplo enteros módulo p con $p > n$). Es de hacer notar que el predicado **par** (utilizado en la solución 1) no tiene sentido utilizando los enteros módulo p con p impar.

Con el axioma $par(n) = par(n \bmod p)$ el universo de modelos es vacío.

La solución 3 depende de la propiedad que adquieren los soldados en el tiempo: están vivos o muertos. La complejidad del algoritmo depende escencialmente de la operación de buscar un vecino vivo. Buscando reducir ésta complejidad encontramos el esquema directriz de la solución 4.

Solución 4

Vamos a basar ésta solución en el tipo abstracto anillo, cuya talla es decreciente en la medida en que los soldados sean eliminados. Se está incluyendo la operación de eliminación de los soldados.

Informalmente este algorítmo se basa en la eliminación de los soldados hasta que quede un solo soldado en el cículo. La determinación de esta condición de terminación puede hacerse de dos formas:

- Introduciendo la función talla del círculo y deteniendo la evaluación cuando esta talla sea unitaria.
- Si un elemento es igual a su sucesor, implica que el anillo es unitario. Debe tenerse en cuenta que ésta propiedad sólo es válida cuando en el anillo no hay elementos repetidos (es una realización de conjuntos).

IMPLEMENTACION

0 $J_2(A : anillo)$
1 si A es vaci'o $J_2 = 0$

```
2 sino
3   Candidato := PRIMERO(A)
4   hacer siempre
5       B := Candidato
6       Candidato := sucesor (Candidato)
7       si Candidato ≠ B entonces
8           eliminar Candidato,
9           declarando como
10          Candidato a su sucesor
11      sino exit
12  J2 := B
12  fin hacer
```

2.4 Propiedades derivables de la solución 4

La condición de terminación de esta implementación es la de obtener un anillo unitario. Vemos que esta solución permite, mediante transformaciones, modificarse para ser solución de $J_r(n)$, ya que basta cambiar la estrategia de eliminación (línea 6):

6' $candidato := sucesor^{r-1}(candidato)$

Esto hace fácil la evolución de la solución a una solución del problema más general.

Hemos mostrado una variedad de soluciones del problema de Josephus.

3 Conclusiones

Hemos presentado una función que se puede expresar informalmente en forma simple cuya implementaciónes varían en función del modelo utilizado para la presentación de los datos. Cada una de implementaciones es esencialmente diferente, dependiendo del modelo del cuál importa los operadores. Los modelos asociados corresponden a álgebras no isomorfas.

Las diferencias más resaltantes desde el punto de vista computacional son la velocidad de cómputo y la memoria requerida para la computación. Estas

implementaciones pueden presentarse como especificaciones, y cada una de ellas puede estar asociada a diferentes modelos.

Como elementos aislados, no se puede jerarquizar estas soluciones, dado que esa jerarquización depende de la arquitectura donde corra la aplicación y de los restantes operadores que sean usados en la aplicación.

Sin embargo la información obtenida durante las descripciones en modelos particulares puede contribuir a la reusabilidad del software desarrollado y para el test de desarrollos futuros. Una versión extendida de éste trabajo se encuentra en [ZOL].

References

[KNU2] R. Graham, D. Knuth, O.Patashnik Concrete Mathematics. A Foundation for Computer Science Addison Wesley Pub. Co. 1989.

[GAU] M. Bidoit, M.C. Gaudel PLUSS: Proposition pour un langage d'especification structurées Reporte interno Universidad de París Sud. 1985.

[ZOL] C.Zoltan Sobre especificaciones y modelos: un ejemplo Reporte N IC 1990-001 Departamento de Computación, Universidad Simón Bolívar Mayo 1990.

SYSTEMATIC PROGRAM DEVELOPMENT FROM ABSTRACT DATA TYPE DECOMPOSITION. A CASE STUDY: THE TOPOLOGICAL SORT ON ORIENTED GRAPHS.

Francis Losavio, Alfredo Matteo
Centro de Ingeniería de Software Y Sistemas ISYS, Facultad de Ciencias, Universidad Central de Venezuela.
Ap. 47002, Los Chaguaramos 1041-A, Caracas, Venezuela.

Françoise Schlienger
Laboratoire de Recherche en Informatique L.R.I., Université Paris-Sud,
Bât.490, Orsay cedex 91405, Orsay, Francia

ABSTRACT

A methodology is presented for automatic construction of algorithms, involving decomposition of an Abstract Data Type (ADT) which is formally specified as an algebraic ADT. As a case study illustrating the proposed methodology, we have selected to construct a program corresponding to the topological sort problem, for an oriented graph data type, which is algebraically specified using a formal specification language. An Ada program is generated following our methodolgy. Early handling of exceptional situations is used as a major programming style. The *program unit* implementing the ADT, a *decomposition scheme* for representing common algorithms on the data type structure and an appropriated *construction strategy* depending on the nature of the problem, are the main elements used in this program construction process.

Our major goal is to show the systematic development of classical algorithms on commonly used structured data types (array, tree, graph, ...) and the derivation of the corresponding program, within an assisted program construction context.

1. INTRODUCTION.

This work presents a methodology [Gre 84], [Los 85], [Los] for automatic construction of algorithms involving decomposition of an Abstract Data Type (ADT) which is formally specified as an algebraic ADT. A case study will illustrate the proposed methodology; for

this purpose we have selected to solve the topological sort classical problem. An oriented graph data type is algebraically specified using the PLUSS [Cap 87], [Bid 89] specification language. Our methodology is independent from the language chosen for implementing the algorithm, provided the language holds ADT and exception handling features. Since Ada provides these features, we have chosen it as our target language. An Ada program is then generated following the methodolgy. Early handling of exceptional situations is used as a major programming style. The language *program unit* implementing the ADT, a *decomposition scheme* for repre ing common algorithms on the data type structure and an appropriated *construction strategy* depending on the nature of the problem, are the main elements used in this program construction process.

Our major goal is to show the systematic development of classical algorithms on commonly used structured data types (array, tree, graph, ...) and the derivation of the corresponding program implementing them, within an assisted program construction context.

This paper is structured in two main parts. The program construction methodology is presented in the first place, focousing on its context, antecedents, actual development, notations and terminology used. The second part is devoted to the description of the case study: the problem definition and the corresponding program obtained applying the methodology. An Appendix contains the oriented graph algebraic specification and the corresponding Ada package implementing the oriented graph ADT.

2. METHODOLOGY.

Within the field of automatic programming, the assisted program construction approach (where program generation is done with the user intervention) is the context of our approach. Richard Waters [R&W 88] describes several techniques employed in problem-solving, one of these being the so called *inspection methods*

based on *clichés*. It is of general agreement that human programmers think seldom in terms of primitive constructs such as assignments and tests, but rather in terms of combinations of elements (a cliché) of familiar concepts. The central feature of inspection methods is the codification and use of clichés, which is constitued of three parts: - a *skeleton* that is present in every occurrence of the cliché, - some *roles*, whose contents vary from one occurrence to the next, - some *constraints* on what can fill the roles. There may be different kinds of clichés; we are particularly interested in *algorithmic clichés*, which belong to a specialized programming area. The "knowledge" of these clichés is represented by common algorithms (such as counting, searching and sorting) and by common data structures (array, sequences, etc.). When used in the assisted construction approach, inspection methods have the important advantage of constituing a shared vocabulary between the system and the user, which seems a natural and attractive medium of communication .

Waters'idea of algorithmic clichés is shared by our proposed assisted program construction method. Using the above terminology, "the knowledge" of our clichés is composed by:

1. The decomposition scheme , which corresponds to a particular way of processing the data structure. The intuitive notion of decomposition of a data type means to consider its data structure from an algorithmic point of view, separated into different parts or blocks, each one of these being constitued by a set of operations defined in the ADT signature. These parts must contain selector operations and at least one decomposition operation, that is to say, a destructor operation for the obtention of the objects of the structure. The algorithm is reflected in the way in which the operations, constituing the different blocks, are placed in order to obtain these objects. There will be two kinds of blocks: BLOCK_SEL, including a selector operation and BLOCK_DES, including a destructor operation. For example, in order to algorithmically deal with a sequence, we may require to separate its first element and process it accordingly, then recursively,

we apply the same reasoning for the rest of the sequence. Consequently, we consider this data structure splitted into two objects: the first element and the sequence constitued by all the elements, excepting for the first one.

An *abstract decomposition scheme* (*ADS*) [Los] is a general formulation for the above decomposition notion, allowing the expression of the algorithmic structure of a data type, by means of the operations on the type, which is algebraically specified.

The schemes we are dealing with are *concrete schemes*, in the sense that they are ADS, applied on a particular data type; they are also *generic schemes*, that is to say, parameterized by the same parameter as the corresponding generic abstract data type.

2. The construction strategy, which is a program scheme or algorithm depending on the nature of the problem (for example counting or searching) we are dealing with. The construction strategy relates the problem we are trying to build, which we will call the *Problem of Interest* (*PI*) and the decomposition scheme, by appropriate combinations of the input and output variables of PI and the output variables of the involved decomposition scheme. The construction strategies also greatly contribute to the structuring of the resulting algorithm, organizing the different blocks constituing the decomposition schemes. Several strategies are present in our methodology: COMBINE, TEST, TEST-COMBINE, ESCAPE [Los] . Each strategy is in general oriented to a particular kind of problem: COMBINE for counting, TEST and ESCAPE for searching, TEST-COMBINE for sorting. The COMBINE strategy, which produces an algorithm in which all the blocks are sequentially organized, will be discussed in details.

In the next section we will detail the program development methodology.

2.1 The program construction process.

The proposed program construction process follows the assisted approach. If we place ourselves in a user-system context, the general idea of the process is the following: first, a problem (PI) is proposed to the system as a function header, expressed in the

target language. The kind of PI and the data structure must be identified by the user. A convenient decomposition scheme is then selected by the user and the system applies (translates) it to the data structure, producing a program skeleton, "instanciated" with the PI variables. This skeleton contains various continuation points which have to be filled with the user assistance, in order to solve the constraints imposed by the selection of an appropiate construction strategy (It is clear that the user has to "master" the construction method, in the sense that he must have some knowlege on the strategy to be used for a particular PI). These constraints are the renaming of new variables and functions and the solving of the possible new sub-problems (some of the introduced new functions). In this way, a dialog system-user is established, until the program is completed. The program construction process is quasi-independent from the target language used (the "quasi" stay for the organization of the exception handling). We have selected Ada as our target language because of the similarity of the Ada generic package with the PLUSS generic SPEC construct, besides the characteristics of modularity and exception handling.

The resulting executable program systematically constructed following this methodology may need minor optimizations, such us eliminating unnecessary intermediate variables or unused exception handlers.

In order to assistedly construct a program corresponding to a given problem PI, the following information is required:

a. A **Specification Library** containing commonly used abstract data types, expressed in a formal specification language (the PLUSS specification language in our case). We will be dealing with *generic abstract data types*. The algebraic specification of the oriented graph data type, witten in PLUSS is shown in the Appendix.

b. A **Program Library** containing the program units implementing the algebraically specified ADT, written in a target language holding abstraction and exception handling features (the Ada language in our case; the Program Library will be called the Ada

Library. The program units contained will be basically Ada generic packages and application programs). The generic package OR_GRAPH, implementing the PLUSS generic specification SPEC OR_GRAPH, may be seen in the Appendix.

c. A **Decomposition Scheme Library** containing the Generic Concrete Decomposition Schemes (GCDS) for the abstract data types present in the Specification Library.

As an example of a GCDS, the oriented graph data type generic concrete decomposition scheme, will be shown below:

```
SCHEME SH1-GRAPH [NODE]
(g : graph --> n : node; g1 : graph)
begin
  n <-1- obtain_node(g);
  [1]
exception
  1 ---> no_node
end
begin
  g1 <-2- rest_of_graph(n,g);
  [2]
end
END SH1-GRAPH
```

Notice that the formal parameter is NODE, and that the exception case is *no_node*, which may be raised by the operation *obtain_node* applied on an empty graph; *rest_of_graph* is a total operation, hence it doesn't raise an exception. The numbers in [ ] indicate a continuation point, where the construction strategy has to be applied.

The GCDS is structered in a header, containing the so called input variable (g:graph) and output variables (n:node, g1:graph) of the scheme and the different blocks, one for each decomposition scheme output variable.

2.2 The construction strategies.

This section describes the application of the construction strategy to a decomposition scheme. We have selected the COMBINE strategy, as an example of this point. Similar description of the other strategies may be seen in [Los].

Application of the COMBINE strategy.

The application of this strategy to a decomposition scheme allows the computations of intermediate results, which will be denoted by R_r, $1 \le r \le k$, k number of blocks of the GCDS, from the set of output variables of the scheme. Each R_r depends entirely from the block where the result is computed, including the results computed in the exception handlers associated to the block. Each R_r is computed by a function F_r whose arguments are the output variables of the scheme and the input variables of PI. In particular, for each F_r, the arguments are constitued by the variable e_i, $1 \le i \le m$ or S_j, $1 \le j \le n$, and all the output variables of the scheme. When variable S_j is present among the arguments of F_r is a recursive call to problem PI. If a function F_r raises exception EX_F_r, its handler is added to the already existing exception handler at the end of the block. At the end of the program body, a new function, called OP-COMBINE will compute the combination of the R_r intermediate results. If this operation raises an exception its handler is added at the end of the program body. The COMBINE strategy is well suited for solving counting type problems. The algorithm shown below is obtained:

The algorithm.

```
function PI (S : TS; I_1,...,I_q) return T is
-- local variable declarations
R_1 : T,... , R_m : T, R_m+1 : TS,...,R_K : TS; X_1 : T_1,...,X_k : T_k; S_1,...,S_n : TS;
e_1,...,e_m : TE;
begin -- body of PI
      begin                                  -- block BLOCK_SEL_1
      X_1 <- OP_PREL_1
      e_1 <- SEL_1(S,Z_1)
      R_1 := F_1(e_1,I_1,...,I_q);
      exception
       when EX_OP_PREL_1 => return F_11(e_1,I_1,...,I_q);
       when EX_SEL_1 => return F_12(e_1,I_1,...,I_q);
       when EX_F_1 => return F_13(e_1,I_1,...,I_q);
      end;                                   -- block BLOCK_SEL_1
      ... ... ...
```

```
begin                                   -- block BLOCK_SEL_m
I_m  <- OP_PREL_m
e_m  <- SEL_m(S,Z_m)
R_m  = F_m(e_m,I_1,...,I_q);
exception
 when EX_OP_PREL_m => return F_m1(e_m,I_1,...,I_q);
 when EX_SEL_m => return F_m2(e_m,I_1,...,I_q);
 when EX_F_m => return F_m3(e_m,I_1,...,I_q);
end;                                    -- block BLOCK_SEL_m
```

```
begin                                   -- block  BLOCK_DES_1
I_m+1  <- OP_PREL_m+1
S_1  <- DES_1(S,Z_m+1)
R_m+1  := PI(S_1,I_1,...,I_q);
exception
 when EX_OP_PREL_m+1 =>return F_(m+1)1(S_1,I_1,...,I_q);
 when EX_DES_1 => return F_(m+1)2(S_1,I_1,...,I_q);
end;                                    -- block  BLOCK_DES_1
```

. . . . . .

```
begin                                   -- block BLOCK_DES_n
I_k  <- OP_PREL_k
S_n <- DES_k(S,Z_k)
R_k  := PI(S_n,I_1,...,I_q);
exception
 when EX_OP_PREL_k => return F_k1(S_n,I_1,...,I_q);
 when EX_DES_n => return F_k2(S_n,I_1,...,I_q);
end;                                    -- block BLOCK_DES_n
return OP-COMBINE(R_1,...,R_k);
exception
  when EX_OP_COMBINE =>  ... ... ... [1]
end PI;
```

Notice that [1] is a continuation point that must be filled by the user with an adequate action. Notice also that the algorithm resulting from the application of the COMBINE strategy is written in an Ada like notation. For simplification, OP_PREL and SEL denote sets of operations; EX_OP_PREL and EX_SEL denote the corresponding sets of exceptions; the symbol <- means that an operation (respectively each operation in case of sets) assign a value to a local variable. In the next section this strategy will be used in the construction of the topological sort program.

3. CASE STUDY: THE TOPOLOGICAL SORT.

Our Problem of Interest, PI, is the following: build a sequence s containing all the nodes of an oriented graph g, in such a way that no node appears before one of its predecessors (PI is called a topological sort). The pre-condition for PI is the requirement that g must be without circuits. The post-condition is a partial ordering on the nodes of g.

The Ada program shown below has been produced accordingly to our methodology:

```
... .... ...

function topol_sort(g:graph) return seq is
n : integer; g1 : graph;
s : seq;
begin
 g1 := g; -- necessary because rest_of_graph modifies graph g
 begin   -- block SEL1
   n := obtain_node(g1);
   exception
     when no_node => return empty;
 end;
 begin   -- block DES1
   rest_of_graph (n,g1);
   s := topol_sort (g1);
 end;
 return add(n,s); -- OP_COMBINE
end topol_sort;

... .... ....
```

Notice that scheme SH1-GRAPH has been applied, and the COMBINE strategy is responsible for constructing the resulting sequence, adding the nodes with negative degree 0, obtained in each iteration as a partial result, by the operation add(n,s), placed at the end of function topol_sort. The operations on the graph data type are implemented in the ORIENTED_GRAPH package shown in the Appendix. The add operation belongs to the package implementing the SEQUENCE data type, which may be seen in [Los]. The algorithm stops when the graph is empty, raising exception no_node by operation obtain_node. We have used here COMBINE, which is a strategy adapted for solving counting problems, instead of TEST-COMBINE, suited for solving sorting problems, because the test on the node having negative degree 0 is implicit in the

obtain_node operation. We point out that this program and all the Ada code [DOD 83] shown in this paper have been tested on the Alsys Ada Compiler of the L.R.I., Université de Paris-Sud, Orsay, France.

The M-APEX protoype Ada programming environment [Los 89], built by the ISYS research center at the Faculty of Science, U.C.V., Caracas, Venezuela, contains the APA (Automatic Programming in Ada) tool [G&V 91], [A&H 91], which implements the program construction methodology presented in this paper

3. CONCLUSION.

The major goal of this paper is to introduce a systematic program construction methodology. It is clear that the algorithms automatically produced may not be optimal ones, in the sense that they may need further optimizations; also other representations implementing the ADT may be selected, in order to obtain more performant algorithms. Our present research focuses on how to "capture" the users experience on problem solving, in order to leave to the system the decison on the selection of the construction strategy, according to a given type of problem. We think also that the construction strategies may be further enriched by several variants. This research will drive us in a near future, towards an expert system, within an assisted program construction context, following this methodology.

4. ACKNOWLEDGMENT.

The authors are gratley indebted to Michel Bidoit, who has followed very closely this research from the beginning, for his pertinent advice and always frutiful remarks and they also thank Oscar Ordaz for his useful comments on the oriented graph ADT.

5. REFERENCES.

[G&V 91] ANES L., HERNANDEZ J.
"Un sistema de construccion automatica de programas Ada: impelemntacion de los moduios de construcción.

Trabajo especial de Grado para la Licenciatura en Computación, UCV. Marzo 91.

[Bid 89] BIDOIT M.
Pluss, un langage pour le développement de specifications algebriques modulaires", Thèse d'Etat, Université de Paris-Sud, Orsay, May 1989.

[Cap 87] CAPY F.
ASSPEGIQUE: un environnement d'exceptions ... .
Thèse de 3ème cycle, Université Paris-Sud, Orsay 1987.

[DOD 83] DEPARTMENT OF DEFENSE OF THE USA.
Reference Manual for the Ada Programming Language
ANSI/MIL-STD 1815-A, January 1983

[Gre 84] GRESSE C.
"Contribution à la programmation automatique. CATY : un système de construction assistée de programmes". Thèse d' Etat, Université de Paris-Sud, Orsay, March 1984.

[G&V 91] GONZALEZ M., VELOZ, C.
"Un sistema de construcción automática de programas Ada: impelemntación de los módulos de interfaz.
Trabajo especial de Grado para la Licenciatura en Computación, UCV. Marzo 91

[Los] LOSAVIO F.
"Dérivation systématique des programmes Ada comportant des traitements d'exception à partir des spécifications algébriques de types de données"
Manuscript for Doctoral Dissertation, Université Paris-Sud, Orsay.

[Los 89] LOSAVIO F.
"Towards the Construction of Interactive Working Environments: A prototype Ada Environment"
Proceedings of the ICNTSSD'89, Caracas, November 1989.

[Los 85] LOSAVIO F.
"Construction assistée de programmes Ada fondée sur la prise en compte d'exceptions".
Thèse de 3ème cycle, Université de Paris-Sud, Orsay, November 1985.

[R&W 88] RICH C, WATERS R.
Automatic Programming. Myths and Prospects.
IEEE Computer 21(8)40-51 August 88.

APPENDIX.

Algebraic specification of the graph data type.

Many different real-life problems or situations (communication networks, tasks scheduling, hierarchy of specification modules ...) may be modeled by a graph. In order

to give a better description of the graph algebraic specification, we will introduce some notations and definitions.

An *oriented graph* g is a pair $\langle S,A \rangle$, where S is a finite set of elements called *nodes* and A is a finite set of ordered pairs called *arcs*. A *non oriented graph* g is a pair $\langle S,A \rangle$, where S is a finite set of nodes and A is a finite set of pairs of nodes called *edges*. In an oriented (or non oriented) graph g, a *path* (or *chain*) of length l is a sequence of l+1 nodes $(n_0,n_1,...,n_l)$, such that for all i, $0 \leq i \leq l-1$, $n_i \rightarrow n_{i+1}$ is an arc (or edge) of g. In an oriented (or non oriented) graph g, a path (or chain) $(n_0, n_1, ..., n_l)$, where all the arcs (or edges) are pairwise different, and such that the initial and terminal nodes of its path (or chain) are equal, is called a *circuit* (or *cycle*). An oriented (or non oriented) graph is called *simple*, when it has no loops (i.e. an arc (edge) with the same extremities) nor multiple arcs (or edges) (i.e. more than one arc with the same orientation (or edge), between two nodes).

In what follows we will consider graphs that are *simple* and without circuits.

We will present here an algebraic specification for oriented graphs, **SPEC** OR_GRAPH, parameterized by the NODE specification, which is not shown here for abbreviating our text. The specified operations are the followings :

The generators are *empty* to initialize a graph, *add_node* to construct a graph from one node and a graph and *add_arc* to construct a graph from an arc and a graph.

The observers are *empty_graph*, which tests if the graph is empty, *exist_node* to test if a node is present in the graph, *exist_arc* to test if an arc is present in the graph, d^- which returns the negative degree (the number of incident nodes) of a node, *nb_nodes* which returns the number of nodes of a graph and *predi* which, given a node and an integer i, returns the i-th predecessor of the node. We adopt the convention that the predecessors of one node are numbered in an increasing order.

The selector is *obtain_node*, to return a node such that $d^-(n,g) = 0$. Note that this is always possible because, in every graph g without circuits, there is always a node n

such that $d^{\circ}(n,g)=0$. The destructor is *rest_of_graph*, to return the graph without a node.

The exception cases are : *no_node*: corresponds to the search for a node, when the graph is empty. *no_graph*: corresponds to the addition of an arc to non existing nodes. *parallel_arc*: corresponds to the addition of an already existing arc. *loop*: is the adition of an arc with the same initial and terminal nodes. *existing_node*: is the addition of an already existing node. *no_pred*: a predecessor of a node cannot be greater than the number of incidents nodes or 0 cannot be the predecessor of a node *no_d°*: it occurs when the graph is empty.

SPEC : OR_GRAPH [NODE]
USE : BOOL, INTEGER
SORTS : graph
GENERATED BY :

empty :	-> graph
add_node :	node graph -> graph
add_arc :	node node graph -> graph

OPERATIONS :

empty_graph :	graph -> bool
exist_node:	node graph -> bool
exist_arc :	node node graph -> bool
obtain_node :	graph -> node
rest_of_graph :	node graph -> graph
d° :	node graph -> integer
nb_nodes :	graph -> integer
predi :	node integer graph -> node
pred_aux :	node integer graph -> node

EXCEPTION CASES :

no_node :	empty_graph(g) = true => obtain_node(g)
no_graph :	or(not exist_node(n1,g),not exist_node(n2,g)) => add_arc(n1,n2,g)
parallel_arc :	exist_arc(n1,n2,g) = true => add_arc(n1,n2,g)
loop :	n1 = n2 => add_arc(n1,n2,g)
existing_node :	exist_node(n,g) = true => add_node(n,g)
no_pred :	or(or(i=0,(i>d°(n,g)),not exist_node(n,g))=>pred_aux(n,i,g)
no_d° :	exist_node(n,g) = false => d°(n,g)

OK-AXIOMS :

empty_graph1 :	empty_graph(empty) = true
empty_graph2 :	empty_graph(add_node(n,g)) = false
empty_graph3 :	empty_graph(add_arc(x,n,g)) = false
exist_node1 :	exist_node(n,empty) = false
exist_node2 :	n1 = n2 => exist_node(n1,add_node(n2,g)) = true

exist_node3 : n1≠n2 => exist_node(n1,add_node(n2,g)) = exist_node(n1,g)

exist_node4 : or(n1=n2, n1=n3) => exist_node(n1,add_arc(n2,n3,g)) = true

exist_node5 : and(n1 ≠ n2, n1 ≠ n3) => exist_node(n1,add_arc(n2,n3,g)) = exist_node(n1,g)

exist_arc1 : exist_arc(n1,n2,empty) = false

exist_arc2 : exist_arc(n1,n2,add_node(n3,g)) = exist_arc(n1,n2,g)

exist_arc3 : and(n3 = n1, n4 = n2) => exist_arc(n1,n2,add_arc(n3,n4,g)) = true

exist_arc4 : or(n3 ≠ n1, n4 ≠ n2) => exist_arc(n1,n2,add_arc(n3,n4,g)) = exist_arc(n1,n2,g)

rest_of_graph1 : rest_of_graph(empty) = empty

rest_of_graph2 : n1 = n2 => rest_of_graph(n1,add_node(n2,g)) = g

rest_of_graph3 : n1 ≠ n2 => rest_of_graph(n1,add_node(n2,g)) = add_node(n2,rest_of_graph(n1,g))

rest_of_graph4 : or(n1 = n2, n1 = n3) => rest_of_graph(n1,add_arc(n2,n3,g)) = rest_of_graph(n1,g)

rest_of_graph5 : and(n1 ≠ n2, n1 ≠ n3) => rest_of_graph(n1,add_arc(n2,n3,g)) = add_arc(n2,n3,rest_of_graph(n1,g))

obtain_node : $\overset{\cdot}{d}^{-}$(obtain_node(g))= 0

$\overset{\cdot}{d}^{-}$1 : n1 = n2 => $\overset{\cdot}{d}^{-}$(n1,add_node(n2,g)) = 0

$\overset{\cdot}{d}^{-}$2 : n1≠ n2 => $\overset{\cdot}{d}^{-}$(n1,add_node(n2,g)) = $\overset{\cdot}{d}^{-}$(n1,g)

$\overset{\cdot}{d}^{-}$3 : n1 = n2 => $\overset{\cdot}{d}^{-}$(n1,add_arc(n2,n3,g)) = $\overset{\cdot}{d}^{-}$(n1,g)

$\overset{\cdot}{d}^{-}$4 : n1 = n3 => $\overset{\cdot}{d}^{-}$(n1,add_arc(n2,n3,g)) = $\overset{\cdot}{d}^{-}$(n1,g) +1

$\overset{\cdot}{d}^{-}$5 : and(n1 ≠ n2, n1 ≠ n3) => $\overset{\cdot}{d}^{-}$(n1,add_arc(n2,n3,g)) = $\overset{\cdot}{d}^{-}$(n1,g)

nb_nodes1 : nb_nodes(empty) = 0

nb_nodes2 : nb_nodes(add_node(n,g)) = nb_nodes(g) + 1

nb_nodes3 : nb_nodes(add_arc(n1,n2,g)) = nb_nodes(g)

pred_aux1 : n1 ≠ n2 => pred_aux(n1,k,add_node(n2,g)) = pred_aux(n1,k,g)

pred_aux2 : and(n1 = n3, k=1) => pred_aux(n1,k,add_arc(n2,n3,g)=n2

pred_aux3 : and(n1 = n3, 1<k≤$\overset{\cdot}{d}^{-}$(n1,g) = > pred_aux(n1,k,add_arc(n2,n3,g)) = pred_aux(n1,pred(k),g)

pred_aux4 : n1 ≠ n3 = > pred_aux(n1,k,add_arc(n2,n3,g)) = pred_aux(n1,k,g)

predi : predi(n,i,g) = pred_aux(n,$\overset{\cdot}{d}^{-}$(n,g)-i+1,g)

WHERE :

n,n1,n2,n3,n4 : node

i,k : integer

g : graph

END OR_GRAPH

Note that this specification is not complete; the implementation of the operation *obtain_node* is left to the specifier's choice.

In order to obtain the specification of a non oriented graph, it is sufficient to change in the OR_GRAPH specification *arc* by *edge* and d^+ by d° ($d^\circ(x,g) = d^+(x,g) + d^-(x,g)$, with x, y nodes of g). Moreover, for every nodes x, y of g, we have : *exist_edge(x,y,g)* => *exist_edge(y,x,g)*.

The graph package.

The implementation given corresponds to the oriented graph data type. The representation is the usual adjacency list. The graph is implemented as an array of pointers to the list of adjacent nodes. We use the *p_array* type of the *sp_array* package, not shown here (see [Los]), for storing the pointers to the adjacency list of the nodes.

```
-- generic package for the graph data type
--
with text_io; use text_io;

with sp_array;

generic
 type node is private;

package oriented_graph is

type inter is range 0..20;

no_node, no_graph, no_arc, existing_node, parallel_arc, loop_ing,
no_pred, no_deg, too_big : exception;

type graph is private;

function empty return graph;
function empty_graph (g : graph) return boolean;
function exist_node (n : node; g : graph) return boolean;
function exist_arc (x, n : node; g : graph) return boolean;
function nb_nodes (g : graph) return integer;
procedure add_node (n:node; g : in out graph);
procedure add_arc (x, n : node; g : in out graph);
procedure rest_of_graph (n : node; g : in out graph);
function neg_degree (n : node; g : graph) return integer;
function obtain_node (g : graph) return node;
function pred (n : node; pi : integer; g : graph) return node;
```

```
private

type struct;

type ptr is access struct;
 type struct is
      record
        no : node;
        next : ptr;
      end record;

package tab_ptr is new sp_array (index => inter, elem => ptr);
use tab_ptr;

-- the graph is implemented as an array of pointers to the list
-- of adjacent nodes

package ind_io is new integer_io(inter);
use ind_io;

package int_io is new integer_io(integer);
use int_io;

type gra is new tab_ptr.p_array;

type graph is
     record
       occ : inter;
       graph_tab : gra;
     end record;

package tab_int is new sp_array (index => inter, elem => integer);

-- the table of negative degrees is implemented as an array of integers

type tab_degree is new tab_int.p_array;

end oriented_graph;

package body oriented_graph is

function empty return graph is
g : graph;

begin
 g.occ := 0;
 g.graph_tab := init(0,20);
 return g;
end empty;

function empty_graph (g:graph) return boolean is

begin
 if g.occ = 0  then return true;
```

```
          else return false;
  end if;
end empty_graph;

function exist_node (n:node; g:graph) return boolean is
i : inter;

begin
   if not empty_graph(g) then
   for i in lwb(g.graph_tab)+1..g.occ loop
    if n = acces(g.graph_tab,i).no then return true; end if;
   end loop;
   end if;
   return false;
end exist_node;

function exist_arc (x, n:node; g:graph) return boolean is
v : ptr;
i : inter;

begin
 if empty_graph(g) then return false; end if;

   for i in lwb(g.graph_tab)+1..g.occ loop
    if x = acces(g.graph_tab,i).no then
     begin
      v := acces(g.graph_tab,i);
      while v /= null loop
       if n = v.no then return true;
             else v := v.next;
       end if;
      end loop;
      return false;
     end;
    end if;
   end loop;

end exist_arc;

function nb_nodes (g:graph) return integer is

begin
 return inter'pos(g.occ);
end nb_nodes;

procedure add_node (n:node; g:in out graph) is
ub : inter;
v : ptr;
-- the node is added at the end of the array

begin
   if exist_node(n,g) then raise existing_node; end if;
   g.occ := g.occ+1;
   v := new struct;
```

```
   v.no := n;
   v.next := null;
   assign(g.graph_tab,g.occ,v);
end add_node;

procedure add_arc (x,n:node; g:in out graph) is
i : inter;
v, v1 : ptr;

begin
   if empty_graph(g) then raise no_graph; end if;
   if x = n then raise loop_ing; end if;
   if exist_arc(x,n,g) then raise parallel_arc; end if;
   if not exist_node(x,g) then raise no_arc; end if;
   if not exist_node(n,g) then raise no_arc; end if;
   for i in lwb(g.graph_tab)+1..g.occ loop
    if x = acces(g.graph_tab,i).no then
     begin
      v := acces(g.graph_tab,i);
      while v.next /= null loop
       v := v.next;
      end loop;
      v1 := new struct;
      v1.no := n;
      v1.next := null;
      v.next:=v1;
      -- the successor is added at the end of the list
     end;
    end if;
   end loop;
end add_arc;

procedure rest_of_graph (n:node; g:in out graph) is
i,k,j : inter;
v : ptr;

begin
   if not empty_graph(g) then
    if exist_node(n,g) then
     begin
      for i in lwb(g.graph_tab)+1..g.occ loop
       if n = acces(g.graph_tab,i).no then
        begin
         k := i;
         -- k contains the index of n in g
         v := acces(g.graph_tab,i).next; -- v points on the successor list
         if v /= null then
          begin
          while v /= null  loop
           if v.no /= n then
                    v := v.next;
                else
                 begin
                   put("v.no ");new_line;
```

```
                        if v.next.next /= null then
                                v.next := v.next.next;
                        -- the element is deleted
                                else
                        v := v.next;
                        end if;
                        exception
                         when constraint_error => v.next := null;
                       end;
              end if;
             end loop;
            end;
           end if;
          end;
         end if;
      end loop;
      -- actualization of the array
      g.occ := g.occ-1;
      for j in k..g.occ loop
       assign(g.graph_tab,j,acces(g.graph_tab,j+1));
      end loop;
      sub_array(g.graph_tab,lwb(g.graph_tab),g.occ);
     end;
    end if;
   end if;
   exception
    when illegal_sub_array => raise no_graph;
end rest_of_graph;


function neg_degree(n:node; g:graph) return integer is
n_deg : tab_degree;
cpt : integer;
v : ptr;
i : inter;

begin
 if empty_graph(g) then raise no_deg; end if;

 cpt := 0;

 for i in lwb(g.graph_tab)+1..g.occ loop
  v := acces(g.graph_tab,i);
  v := v.next;
  while v /= null loop
   if v.no = n then cpt := cpt +1;
   end if;
   v := v.next;
  end loop;
 end loop;
 return cpt;

end neg_degree;
```

```
function obtain_node (g:graph) return node is

-- the first element of the array whose negative degree is equal 0
-- is taken

begin
 if empty_graph(g) then raise no_node; end if;

 for i in lwb(g.graph_tab)+1..g.occ loop
  if neg_degree(acces(g.graph_tab,i).no,g) = 0 then return acces(g.graph_tab,i).no;
  end if;
 end loop;

end obtain_node;

function pred(n:node; pi:integer; g:graph) return node is
i : inter;
np : integer;
v : ptr;

begin
 if not empty_graph(g) then
  if exist_node(n,g) then
    begin
     if pi > neg_degree(n,g) or pi = 0 then raise no_pred; end if;
     np := 0;
     for i in lwb(g.graph_tab)+1..g.occ loop
      v := acces(g.graph_tab,i);
      v := v.next;
      while v /= null
          loop
           if v.no = n then
              begin
               np := np+1;
               if pi = np then return acces(g.graph_tab,i).no; end if;
              end;
           end if;
           v := v.next;
          end loop;
     end loop;
    end;
  end if;
 else raise no_pred;
 end if;
end pred;

end oriented_graph;
```

CRITERIOS DE SELECCION PARA LAS FORMAS DE REPRESENTACION DEL CONOCIMIENTO

Elena Pérez Miñana, Alejandro Teruel (1)

RESUMEN: El análisis de los conceptos fundamentales asociados a la representación del conocimiento en el computador permiten evidenciar la dependencia que existe entre las formas de representación del conocimiento y los tipos de un área de aplicación. Para alcanzar el objetivo deseado se hizo una revisión general de las distintas formas de representación del conocimiento que se han venido utilizando en el campo de la Inteligencia Artificial. También se realizó un estudio de las diferentes áreas de aplicación de los Sistemas Basados en el Conocimiento (SBC). Estos dos estudios permitieron proponer un conjunto de esquemas de selección que ayudarán a los diseñadores de SBC a identificar las formas de representación más apropiadas para SBCs específicos.

I.-Introducción

El área de SBC ha venido desarrollándose a pasos agigantados en los últimos años. Hoy en día el analista/programador encargado de desarrollar un SBC se encuentra con una multitud de formas de representación del conocimiento (FRC), entre las cuales se incluyen guiones, procedimientos, lógicas de las más variadas especies, encuadres, redes semánticas, reglas de producción y hasta bases de datos relacionales. Cada una de estas formas de representación tiene sus idiosincracias en cuanto a la naturalidad con que logra capturar clases de conocimiento y en cuanto a la facilidad y eficiencia con que se le logra implementar en un computador. Seleccionar una o más de esas FRCs se ha convertido en un problema no trivial, cuyo impacto en el éxito futuro del SBC a desarrollar es crítico.

En el presente trabajo se propone un esquema de selección de FRCs basado en el tipo de conocimientos que se deseen que el SBC maneje. La propuesta deberá tomarse como un rico conjunto de conjeturas cuya comprobación empírica será el sujeto de futuras investigaciones en el área.

El esquema propuesto está basado en una clasificación pragmática de los tipos de conocimiento, de manera que el analista/programador del SBC no requiera de un dominio profundo del campo de la Epistemología.

---

(1)Profesores del Departamento de Computación y Tecnología de la Información de la Universidad Simón Bolívar.

Para ello se ha partido de una clasificación de las áreas en las que se aplican los SBC, basada en los trabajos de Gevarter (1987), Chandrasekaran (1987), Bonnet (1984) y Waterman (1984). La clasificación considerada se muestra en la tabla 1.

ACTIVIDAD	PROBLEMA QUE INTENTA RESOLVER
PERCEPCION	CAPTACION Y PROCESAMIENTO DE LAS SEÑALES RECIBIDAS DEL MUNDO EXTERIOR
PROCESAMIENTO DEL LENGUAJE NATURAL	COMPRENSION Y GENERACION DE FRASES.
CLASIFICACION	SELECCION DE UNA CATEGORIA DE ENTRE UN CONJUNTO PRE-FIJADO DE CATEGORIAS EN BASE A LAS CARACTERISTICAS DEL OBJETO A CLASIFICAR.
DIAGNOSIS	IDENTIFICACION DE LAS CAUSAS DEL MAL FUNCIONAMIENTO DE ALGUN OBJETO
PLANIFICACION	DEFINICION DE UNA SECUENCIA DE ACCIONES QUE DEBE SE-GUIRSE PARA LOGRAR ALGUN OBJETIVO DADO.
SCHEDULING	ORDENACION DE ACTIVIDADES Y RECURSOS PARA OBTENER UNA RESPUESTA EN TIEMPO MINIMO.
DISEÑO	GENERACION DE LA CONFIGURACION DE DISTINTOS TIPOS DE OBJETOS QUE SATISFAGAN UN CONJUNTO DE RESTRICCIONES PREDEFINIDO EXISTEN TIPOS SEGUN EL NIVEL DE CREATIVIDAD
MONITOREO	OBSERVACION CONTINUA DE UN CONJUNTO DE OBJETOS CON CON LA FINALIDAD DE CHEQUEAR QUE LOS MISMOS RESPON-RESPONDEN SIEMPRE DE LA FORMA ESPERADA.
CONTROL DE PROCESOS	CONTROLA EL COMPORTAMIENTO DE UN CONJUNTO DE PRO-CESOS QUE REALIZAN ALGUNA TAREA. SE DIFERENCIA DEL MONITOREO POR LA POSIBILIDAD DE DETERMINAR ACCIONES A APLICARSE A LOS OBJETOS DE LA CALLE.
ENSEÑANZA	SELECCION Y APLICACION DE UNA ESTRATEGIA DE ENSEÑANZA A PARTIR DE PARAMETROS DE EVALUACION QUE DEFINEN EL PERFIL DE UNOS ESTUDIANTES PARA HACERLES ENTENDER ALGUN BLOQUE DE CONOCIMIENTO ESPECIFICO.

Areas de Aplicación de SBCs

Tabla 1

Al llegar a un nivel suficientemente específico de clases de conocimiento es relativamente fácil analizar las bondades y deficiencias de cada FRC para representar la clase considerada. Así por ejemplo, la lógica proposicional es relativamente inaplicable para representar conocimientos operacionales (definidos en la parte III) en los que el tiempo y el medio circundante son factores determinantes, como es el caso de todo el conjunto de partes operativas en

una situación de combate.

El esquema de selección de la(s) FRC es relativamente sencillo y consta de los siguientes pasos:

a.-Identificar el área de aplicación al que pertenece el SBC.

b.-Seleccione los tipos de conocimiento que se deben representar.

c.-Seleccione para cada tipo de conocimiento a representar las FRC más apropiadas.

d.-Intersecte los conjuntos de FRCs hallados en el paso anterior para obtener el FRC más adecuado.

En el último paso pueden ocurrir dos casos de consideración:

d.1.-La intersección es vacía. En este caso puede:

*Bajar el umbral de aceptación de FRCs en el paso (c), o

*Desarrollar un SBC que incluya más de un tipo de FRC.

d.2.-La intersección produce más de un FRC. En este caso se puede:

*Subir el umbral de aceptación de FRCs en el paso c, o

*Escoger alguno de esos FRC en base a otros criterios (e.g. el dinamismo del conocimiento), o

*Desarrollar prototipos de SBC para evaluar la bondad de los FRCs rivales.

El resto del trabajo está fraccionado de la siguiente manera: la Parte II expone la clasificación del conocimiento utilizada, la Parte III esquematiza la bondad de las diferentes formas de representación consideradas para representar los tipos de conocimiento definidos, la Parte IV presenta los conceptos fundamentales aplicados en la definición de los esquemas de selección, la parte V se expone un ejemplo en el que se especifica la forma de evaluación utilizada para obtener los resultados asociados a la planificación y que refleja el proceso de evaluación seguido con todas las AG presentadas en la tabla 1 (Pérez-Miñana 1990), en la parte VI se exponen las conclusiones a las que se llegaron con esta investigación y los posibles desarrollos futuros en este campo.

II.-Clasificación del conocimiento utilizada

Visualizando el conocimiento como un objeto genérico, sin puntualizar si es o no de un tipo en particular, el mismo está compuesto de dos partes fundamentalmente:

a) Un conjunto de hechos.

b) Un grupo de formas para interpretar esos hechos.

El primer elemento es el clave en esta investigación, ya que lo que se busca son esquemas que

permitan tipificar las formas de representación, elemento estructural de la base de conocimientos. El grupo de formas de interpretación se refleja de dos formas en los SBC:

a) La forma en que las unidades básicas de información están relacionadas en la base de conocimientos (los hechos).
b) La máquina de inferencia, otro de los elementos que deben definirse al desarrollar un SBC.

Esta división elemental pone de manifiesto el hecho de que, en el momento de decidir cúal es la forma de representación que más conviene utilizar para un tipo de conocimiento en particular, se deben tomar en consideración no sólo la primera de las partes señaladas, sino también los elementos que influyen en la interpretación de los mismos; sin esto las unidades básicas se convierten en simple información y se pierde la noción de conocimiento, puesto que lo que hace a la información conocimiento es precisamente la posibilidad de interpretación.

Un rasgo adicional del conocimiento que debe tomarse en cuenta, pues de lo contrario se pierde la esencia del mismo, es su dinamismo. Esto puede captarse de dos formas:

a) Soportando el crecimiento y variación del nivel de creencias del conjunto almacenado en la base de conocimientos (no-monotonicidad, incertidumbre).
b) Incluyendo formas de representar el tiempo en todos los elementos incluidos en la base.

Precisando todo lo expuesto, los tipos de conocimiento considerados son:

***Descripción de objetos**: los componentes principales del mundo, deben conocerse a partir de ciertas características (p.e. señales sensoriales) para poder interactuar con ellos. El conjunto de características asociadas a determinados elementos deben permitir, en todo momento puntualizarlo. Considerando que todo objeto constituye un ente distinguible, y que por si mismo resulta ser un sistema, se le puede describir desde las mismas perspectivas que a un sistema, lo cual genera en este tipo de conocimiento una subdivisión adicional, a saber:

a-Descripción cualitativa de objetos: es la más utilizada, por lo menos durante los primeros años de vida de cualquier ser humano. Está asociada directamente a las señales percibidas a través de los sentidos, o a las conceptualizadas para distinguir los objetos imaginarios (e.g. una manzana es dura, tiene una depresión de color blanco, es de color rojo o verde).

b-Descripción funcional de objetos: relación entre entradas y salidas, sin tomar en cuenta la descripción estructural (i.e. interna) del objeto.

Permite responder al tipo de pregunta ¿Qué función ejerce?. Es un tipo de conocimiento empleado al realizar una tarea de monitoreo por ejemplo, ya que continuamente se analizan si las salidas de los distintos procesos evaluados son las esperadas.

c-Descripción operacional de objetos: describe el mecanismo interno a partir del cual un objeto es capaz de satisfacer la función que efectúa (e.g. al usar una máquina de refrescos, las monedas que se insertan son detectadas y evaluadas por el mecanismo detector de monedas que suma el total al que corresponden para verificar que se ha introducido el cambio correcto y así sucesivamente). Es un tipo de conocimiento requerido para efectuar una diagnosis, por ejemplo.

d-Descripción de manipulación de objetos: describe como usar un objeto, como se comporta sin tomar en consideración su mecanismo interno (e.g. para manejar una máquina de refrescos se deben seguir los siguientes pasos: introducir el cambio señalado, presionar la tecla que contiene el nombre del refresco deseado, esperar a que la máquina seleccione la lata y la entregue a través del conducto de salida).

Además de las características propias de cada elemento según el tipo de descripción que se utilice, cualquiera de los tipos mencionados incluye la relación que entre las partes existe, relación que en el caso de la descripción cualitativa es de pertenencia, en los otros casos se puede considerar de funcionalidad. Este tipo de relaciones permiten generar un conjunto inicial de interpretaciones que, posteriormente, pueden modificarse en presencia de otras relaciones que forman parte del conocimiento y que se describirán a continuación.

***Conocimiento de Interpretación:** permite generar nuevas reglas a partir de las cuales se pueden producir nuevas conclusiones que son importantes en el proceso de resolución de problemas, actividad para la cual el uso del conocimiento es clave. Puede subdividirse de la siguiente forma:

a-Restricciones: en general una restricción puede definirse como un conjunto de condiciones que deben satisfacerse para poder efectuar algún proceso, para ejecutar determinada acción, o para considerar que un elemento pertenece a determinada clase. El último corresponde a las inherentes, son las características asociadas a cualquier elemento y son necesarias para poder afirmar que pertenece a una clase en particular. Los dos primeros comprenden el conjunto de condiciones entre los distintos "objetos" que

interesa considerar, más precisamente, entre los estados de esos objetos.

b-Relación Causa-Efecto: permite determinar los causantes del mal funcionamiento de un objeto, por ejemplo. Se utiliza a menudo en conjunto con la descripción operacional y funcional de objetos. Es un tipo de relación que normalmente asocia objetos de distintas clases (e.g. bacterias, enfermedades).

c-Heurísticas: comprende el conjunto de reglas que cualquier ser humano utiliza en su tarea de resolución de problemas cuando no conoce con exactitud la manera de proceder. Puede ser de tipo genérico (e.g. característica más relevante de una situación) o asociada a un problema en particular (e.g. estrategia de enseñanza).

d-Reglas para modelar objetos: permite a cualquier ser humano integrar todas las señales sensoriales percibidas para armar el modelo mental del objeto que está percibiendo, elemento a partir del cual podrá trabajar en sus procesos cognoscitivos posteriores.

Todos los tipos de conocimiento arriba mencionados se ven afectados por una característica adicional del conocimiento que resulta ser uno de los aspectos más difíciles de capturar en un computador, se hace referencia al dinamismo. Este aspecto se hace evidente a través de los distintos grados de creencia que se asocia a los trozos básicos de alguno de los tipos arriba mencionados y a la variación que éstos sufren en el tiempo.

III.-Bondad de las formas de representación con los tipos de conocimiento

Cada forma de representación del conocimiento (FRC) tiene asociada determinadas características que la hacen particularmente conveniente para almacenar uno o unos tipos de conocimiento en el computador.

Las FRC consideradas en este trabajo fueron: los procedimientos, los triples <O,P,V>, las bases de datos relacionales, las bases de datos semánticas, las redes semánticas, los encuadres, los guiones, las reglas de producción, la lógica de primer orden, la lógica no-monotónica, la lógica modal, la lógica temporal y la lógica difusa.

Excepto por las lógicas no convencionales todas las otras formas de representación funcionan en forma adecuada para distintos tipos de conocimiento, siempre y cuando, los mismos no incorporen un grado de dinamismo que las haga ineficientes y los mecanismos de interpretación asociados sean efectivos. El dinamismo depende en cierta forma del área de aplicación. En ese caso las lógicas no convencionales son las que resultan ser más adecuadas. Se debe señalar que de

todas las formas de representación las reglas de producción han resultado ser bastante atractivas en varias ocasiones debido a que intentan soportar el dinamismo del conocimiento en formas que las otras FRC no consideran.

Por último debe señalarse que el hecho de que una FRC no se considere adecuada no quiere decir que no se pueda utilizar, simplemente se desea indicar la poca utilidad que puede obtenerse al usarla.

En la tabla 2 se presenta el resultado de la evaluación realizada. Las columnas de la tabla corresponden a los tipos de conocimiento y las filas a las FRC. Cada posición contiene un valor en el rango (1,5), que significa: conveniente (5), poco conveniente (4), regular (3), poco útil (2), mala (1). En la tabla se muestran dos valores en cada casilla, el valor de la izquierda indica la conveniencia de representación sin tomar en cuenta el dinamismo del conocimiento, el valor de la derecha representa la conveniencia si se toman en cuenta las características dinámicas del conocimiento. La barra doble divide los tipos de conocimiento de descripción de objetos de los tipos de mecanismos de interpretación.

FRC / TC	DCO	DFO	DMO	DOO	REST	CA-EF	HEUR	RMO
PROCEDIMIENTOS	1/1	3/1	3/1	3/1	1/1	1/1	1/1	4/1
BDA RELACIONAL	3/1	1/1	1/1	1/1	2/1	2/1	1/1	1/1
BDA SEMANTICA	4/2	3/2	2/1	2/1	2/2	3/2	1/1	2/2
TRIPLES<O,P,V>	3/1	1/1	1/1	1/1	1/1	2/1	1/1	1/1
ENCUADRES	4/3	2/2	2/2	2/2	2/2	3/2	1/1	1/1
GUIONES	1/1	3/1	2/1	2/1	2/1	1/1	1/1	1/1
REDES SEMANTICAS	4/2	3/2	2/1	2/1	2/1	3/2	1/1	2/2
REGLAS DE PRODUC.	2/4	2/3	2/3	2/3	4/4	4/4	4/3	2/3
LOGICA DE 1er ORDEN	3/1	2/1	2/1	2/1	4/1	3/1	3/1	1/1
LOGICA NO-MONOT.	2/4	2/3	2/3	2/3	3/4	2/3	3/3	1/2
LOGICA TEMPORAL	1/3	1/3	1/3	1/3	2/4	2/3	2/3	1/2
LOGICA DIFUSA	1/3	1/3	1/3	1/3	2/4	2/3	3/4	1/2
LOGICA MODAL	2/3	2/3	2/3	2/3	3/4	2/3	3/3	1/2

FORMAS DE REPRESENTACION DEL CONOCIMIENTO vs. TIPOS DE CONOCIMIENTO
Tabla 2

IV.-Esquemas de selección

Se han identificado distintos tipos de conocimiento, se ha evaluado la bondad de cada forma de representación para simular cada tipo de conocimiento en

forma aislada. Sin embargo, se debe tomar en cuenta que, ninguno de estos tipos trabaja en forma independiente en la mente humana. Por lo tanto, para poder evaluar en realidad la conveniencia del uso de cada forma, se deben buscar maneras de agruparlos que resulten lógicas dentro de todo el proceso de investigación seguido. Es por esa razón, que se decidió considerar conjuntos de procesos que el ser humano realiza con bastante frecuencia y que se han intentado simular a través de los SBC. Cada grupo se reconoce porque es usado para algún tipo de objetivo en particular, pero un objetivo que no resulte ser demasiado específico como para hacer el análisis inútil. Por ello se utilizaron las actividades genéricas (AG) expuestas en la tabla 1, en la misma tabla se asocia a cada una de ellas una breve definición.

Cualquier ser humano al realizar alguna AG emplea ciertos tipos de conocimiento. El análisis a partir del cual se detectaron los tipos usados sigue la estrategia definida por Chandrasekaran (1987), quien considera que en cualquiera de ellas se deben identificar los siguientes elementos:

a) Los tipos genéricos de información que constituyen las entradas y salidas del sistema.
b) Los trozos básicos de conocimiento del dominio.
c) Los elementos de control de la actividad.

Estos tres elementos permiten definir los componentes de información a partir de los cuales se genera y procesa el conocimiento que cualquier AG requiere para poder ejecutarse adecuadamente.

Los resultados, presentados en la Tabla 3, son de la forma (FRC,tipos de conocimiento) construída asignando a cada posición de la tabla un número entero del conjunto (1, 3, 5, 7, 9, 10) que representa la efectividad de cada FRC para manejar el tipo de conocimiento al que está asociado por la posición de la tabla. Dado que no se puede decir que una FRC no funciona para un tipo en particular sólo se asocian grados de conveniencia, es decir que a una FRC más adecuada para representar un tipo de conocimiento se le asignará uno de los valores del rango mencionado, el grado de efectividad es mayor si el valor asociado es alto, por lo tanto la mínima efectividad está representada por el número 1 y la máxima por el valor 10.

Seguidamente se muestra un ejemplo con el cual se representa, el tipo de razonamiento seguido para la definición de los tipos de conocimiento asociados a cada AG y las FRC que resultan ser más convenientes para construir la base de conocimientos de un SBC definido con el objetivo de realizar dicha actividad.

V.-Ejemplo práctico

La AG planificación, es un proceso de selección a partir del cual se obtienen un conjunto de acciones de un dominio complejo de alternativas, intentando satisfacer los

objetivos del usuario (Gevarter 1987), (Waterman 1984). Ejemplos de sistemas que efectúan esta actividad son: SBC definidos con el objetivo de generar planes para aplicar una serie de reacciones químicas a un grupo de sustancias básicas sintetizando compuestos orgánicos complejos; un sistema de planificación de estrategias de ataque aéreo; un sistema para la definición de planes de ensamblaje mecánico con el uso de robots.

Si se efectúa un análisis del conocimiento empleado en la actividad de planificación, siguiendo la estrategia definida por Chandrasekaran (1986) y los distintos SBC desarrollados para efectuarla, los elementos de información que se deben manejar son los siguientes:

(1) Entradas y salidas del sistema: las entradas comprenden los objetivos que el usuario desea satisfacer. Los mismos pueden ser: un compuesto orgánico complejo (Sowa 1984); la necesidad de destruir determinado objetivo en un ataque aéreo, TATR (Callero 1984); el ensamblaje de algún equipo eléctrico utilizando robots (Chang 1988). Los objetivos son formas de conocimiento que se pueden representar en cualquier dominio como metas, esto es como resultados finales. En cualquier caso, la salida se compone de un conjunto de acciones ordenadas que al aplicarse satisfacen con bastante efectividad los objetivos deseados. El conjunto de acciones es ordenado porque frecuentemente la ejecución de una acción en particular depende de los resultados obtenidos de la ejecución de una operación anterior.

(2) Los trozos básicos de conocimiento que se necesitan para efectuar esta actividad son:

a) El conjunto de acciones contenidas en un plan deben ser efectuadas por algún objeto del dominio, y sobre algún objeto del mismo, por lo tanto, se debe tener conocimiento de todos los objetos que de alguna manera actúan en el mundo; en el caso de un sistema de ataque aéreo, el mismo debe saber las características de los distintos aviones y armamentos que están disponibles, así como, en la medida que sea posible, debe conocer la distribución geográfica de la fuerza enemiga (Callero 1984); si el sistema está dirigido a la planificación de las operaciones de ensamblaje, el mismo debe saber la estructura de las distintas piezas a ensamblar y la relación de dependencia que existe entre una y otra (Chang 1988), además, debe conocer las propiedades y rasgos de cada componente;

b) Las operaciones que pueden efectuarse en el dominio en el que se está desarrollando el SBC; muchas de ellas están bastante bien definidas y forman parte de los principios fundamentales del dominio de trabajo, muchas otras, sin embargo, constituyen acciones que un experto ha definido en el transcurso de sus años de trabajo

(Chang 1988); e.g., en el sistema de ensamblaje este conocimiento comprende todas las operaciones de ensamblaje que puede ejecutar un robot; a nivel de un sistema de acción militar, constituye todas las acciones con las que se pueden mermar las fuerzas del enemigo, las estrategias militares que se aprenden al formar parte de distintos batallones.

(3) La información de control con la que debe contar un SBC para poder planificar efectivamente está constituida por restricciones y heurísticas. Las primeras suceden porque la ejecución de una determinada operación muchas veces imposibilita la aplicación de otra, o hace poco productivos los resultados que se pueden obtener al efectuarla. Estos datos ayudan enormemente a reducir el espacio de búsqueda, pues, el conjunto de operaciones constituye por si solo un dominio bastante grande y la posibilidad de reducirlo en un tiempo corto ayuda mucho a aumentar la eficiencia del sistema. No sólo existen restricciones que se generan automáticamente al presentarse determinada situación, además existen situaciones en las que no conviene aplicar determinadas operaciones, por la poca productividad que se tiene al efectuarlas; la presencia de este tipo de situaciones sugiere la necesidad de incluir las heurísticas en la base de conocimientos, información a partir de la cual se puede decidir cual es el conocimiento que más conviene utilizar.

Los tipos de conocimiento que se acaban de definir permiten satisfacer de una manera bastante completa los requerimientos generales de una tarea de planificación (Chang 1988), (Franklin 1988). Algunos sistemas pueden, adicionalmente, incluir información sobre planes que son muy utilizados en el dominio de aplicación, esto con la finalidad de reducir el tiempo de búsqueda.

Dados los tres componentes de información que debe contemplar el diseñador de un SBC, se puede concluir que los tipos de conocimiento que maneja un individuo al efectuar la tarea de planificación son: descripción funcional y cualitativa de los objetos del dominio incluyendo información sobre la relación que existe entre los mismos en lo que se refiere a los planes posibles; definición de todas las operaciones que pueden efectuarse al desarrollar cualquier plan en el área de aplicación, ésta debe incluir los objetivos finales que pueden lograrse al ejecutarla; por último, una descripción de las restricciones o relaciones de dependencia que existe entre las diferentes operaciones y heurísticas para manejar el proceso de selección. En la tabla (FRC vs tipos de conocimiento de la planificación) se muestra, en forma resumida, la relación que existe entre las distintas FRC y los tipos de conocimiento considerados en

cuanto a efectividad de representación.

T(FRC, TC)	DESCRIP. OBJ. FUNCIONAL, CUALITATIVA	OPERACIONES	RESTRICCIONES/ HEURISTICAS
PROCEDIMIENTOS	5	5	1
BD.RELACIONAL	3	1	1
BD SEMANTICA	9	7	3
ENCUADRES	9	7	1
GUIONES	3	9	1
REGLAS	3	3	9
REDES SEMANT.	9	5	3
T. <O,P,V>	3	3	1
LOGICA 1er Orden	3	3	5
LOGIC.NO-MONOTON	7	7	7
LOGICA DIFUSA	7	5	7
LOGICA TEMPORAL	3	3	3
LOGICA MODAL	5	5	5

FRC VS. CONOCIMIENTOS DE LA ACTIVIDAD DE PLANIFICACION

A continuación se expone el análisis a partir del cual se obtuvieron los resultados de la tabla.

1) Descripción de objetos

La planificación requiere conocer con bastante precisión los elementos sobre los que se debe actuar, o con los que se puede actuar; de los primeros interesa la descripción cualitativa porque a través de ella es que se puede determinar la relación de los objetos con los planes, de los segundos interesa la descripción funcional porque es así que se puede inferir si son útiles para algún plan en particular, no sólo interesan las características particulares de cada elemento, también, si existe algún tipo de relación entre los mismos, el SBC debe estar al tanto de que existe esta asociación, lo cual se logra a través de las restricciones y de las operaciones. Las FRC más adecuadas para este caso son: los encuadres, redes y bases de datos semanticas, lo cual se observa en los siguientes articulos (Gevarter 1987), (Waterman 1986), (Chang 1988), para dominios pequeños los triples <O,P,V> las bases de datos relacionales son suficientes; por ejemplo en el sistema de planificación de las operaciones de ensamblaje que efectúa un robot (Chang 1988), usando encuadres se puede obtener rápidamente la relación de dependencia que existe entre las partes de un equipo, la cual resulta ser básicamente jerárquica, y que a nivel de ensamblaje es importante para poder elaborar el plan de construcción que debe seguir cualquier robot para armar un

equipo. Si se considera el desarrollo del sistema BATTLE (Slagle 1985), (Franklin 1988), la relación que existe entre los objetos no es, como en el caso anterior, de dependencia jerárquica, pues lo que se desea saber es el estado de cada una de las armas, información que se obtiene asignando valores a un conjunto de atributos, conociendo de esta manera la efectividad que tiene cada una de ellas para destruir los distintos armamentos con los que cuentan las fuerzas enemigas; este tipo de relación es de efectividad (capacidad de destrucción) de todos contra todos, luego no puede reflejarse adecuadamente con una estructura jerárquica; pero las redes semánticas y las bases de datos semánticas constituyen estructuras con las que resulta bastante eficiente representar relaciones de este tipo, de hecho en el sistema BATTLE esta información se representa utilizando una variante de las redes semánticas.

Un componente de información, que en algún momento puede resultar importante, es el tiempo, factor éste, muy bien manejado con la lógica temporal.

Los guiones (scripts) y los procedimientos no presentan características que faciliten realmente la representación del mismo en un computador, por lo menos a nivel de la descripción cualitativa de objeto, pero los procedimientos pueden resultar convenientes en la descripción funcional.

2) **Información de las operaciones**

Es el conocimiento de todas las posibles acciones que pueden efectuarse en un dominio de aplicación. Debe incluir, no sólo conocimiento de las acciones en sí, además debe saber, de alguna manera, en qué situaciones puede aplicarlas y cuáles son los resultados que obtendrán luego de ejecutarlas. Una acción no es un elemento aislado, es una forma de interactuar con el mundo exterior, con la finalidad de modificarlo para lograr el objetivo que se persigue; por lo tanto, además de la definición de cada acción, el sistema debe estar al tanto de cómo las mismas interactúan entre sí, qué objetos del dominio pueden ayudar a efectuarla y debe poder determinar la conveniencia de aplicarla en una situación en particular, esta última información depende de una gran cantidad de factores a los que no se les puede asignar un valor con un cien por ciento de seguridad, por lo tanto, requiere manejar esta imprecisión o incertidumbre.

El conjunto de todas las posibles acciones constituye, aún en los dominios más sencillos (un juego de ajedrez por ejemplo), un complejo universo que probablemente no pueda definirse completamente en los dominios extensos, por lo tanto, la solución por la que han optado muchos de los investigadores al desarrollar este tipo de sistemas (Chang 1988), (Slagle 1985), (Duda 1979) consiste en incluir un dominio de operaciones que hayan considerado completo a través de diferentes estudios, y un conjunto de heurísticas,

o factores, que permitan definir una combinación de operaciones cuyos resultados sean los más eficientes.

El uso de procedimientos y guiones (scripts) a nivel de la descripción de las operaciones pueden resultar convenientes, por lo menos en lo que se refiere a las descripciones de las primitivas pero el resto del conjunto sólo puede manejarse con otra FRC. Las bases de datos relacionales y la lógica de primer orden resultan poco convenientes.

La dependencia que hay entre las distintas operaciones se representa con facilidad utilizando las redes semánticas o las bases de datos semánticas. En la primera cada nodo de la misma puede representar un plan, subplan u operación en particular, los arcos conectarían al nodo en cuestión a aquellos nodos que representan las posibles acciones que pueden ejecutarse una vez efectuada la acción correspondiente al nodo padre (Slagle 1985); la información asociada a cada nodo y a cada arco puede resultar ser bastante compleja, porque corresponde no sólo a las características particulares de cada operación (efectos, objetivos que satisface por ejemplo), además debe tener asociados todos aquellos factores que los expertos consideran que deben chequearse, y los mismos deben modificarse cuando corresponda y por las razones que así lo ameriten; las bases de datos semánticas son más flexibles luego más adecuadas.

Junto con la estructura de las operaciones deberá existir otra, bajo la forma de aserciones o predicados lógicos que contendrá todas las heurísticas aplicables incluyendo reglas sobre los nodos de la red en los que cada una debe actuar. Esto se representa con las formas lógicas o las reglas de producción.

Por último, falta considerar los triples <O,P,V>; esta forma de representación es utilizada en el sistema BATTLE (Slagle 1985) como estructura que contiene los trozos de información que deben transmitirse entre los nodos; resulta ser bastante adecuada porque el número de factores que considera importantes son solo 55, y el número de funciones que utiliza para propagar información de un nodo a otro hace que esta información sea bastante manejable usando los triples.

3) **Descripción de las restricciones**

Ayuda principalmente a relacionar todos los elementos del conjunto de conocimiento requerido. Se necesita, fundamentalmente, para resolver conflictos empleando la información contenida en las restricciones, como sucede en el SBC definido por los investigadores Chang y Wee (1988). Las restricciones pueden ser datos asociados a los objetos o las operaciones del dominio de aplicación resultado de la aplicación de determinadas heurísticas definidas por los expertos. Muchas veces una acción en particular es suspendida temporalmente hasta que las variables que tiene

asociadas no toman algún valor. La forma de representación que más naturalmente se adapta para su definición resultan ser las reglas de producción o, las distintas lógicas; cada objeto u operación en su descripción debe contener un bloque que indique qué restricciones deben satisfacerse en un momento en particular, junto con las variables cuyos valores indican precisamente que se ha satisfecho una determinada condición. Estas condiciones puede suceder que no sean las mismas en todos los objetos y que para cada objeto varíen de una situación a otra, luego, resulta más conveniente definir las condiciones bajo una forma lógica con la cual se refleje esta situación, que es lo que sucede con las lógicas modal y no-monotónica, aunque el concepto de mundos posibles de la lógica modal parece ser más apropiado.

Las otras formas de representación existentes no están conceptualizadas con el objetivo de definir condiciones, de la forma descrita luego no pueden considerarse como formas de representación adecuadas.

En el trabajo "Esquemas de selección para las formas de representación del conocimiento" (Pérez-Miñana 1990) se desarrollaron análisis, siguiendo el mismo procedimiento presentado, para el resto de las AG consideradas. Debe señalarse que a la AG control de procesos no se le asoció una tabla de resultados como a las otras actividades, pues se consideró, que la misma se realiza indirectamente dentro de cualquiera de las AG mencionadas, por lo que los tipos de conocimiento requeridos de alguna forma están implícitos en las otras tablas de resultados.

VI.-Conclusiones

En este punto se presentan algunas conclusiones obtenidas con esta investigación y también, posibles investigaciones futuras sobre este tema:

(1) La clasificación del conocimiento presentada en la parte II de este informe resultó conveniente para la definición de los esquemas de selección.

(2) El conjunto de actividades denominadas AG permiten hacer una clasificación bastante completa y precisa de todos los SBC desarrollados hasta los momentos. Esto se afirma ya que para cada actividad presentada en la tabla se logró definir los tipos de conocimiento que se requieren para efectuarla, y a este nivel, es posible asociar a cada uno de ellos una o varias formas de representación más convenientes.

(5) Uno de los problemas, por los cuales no se puede conseguir una simulación realmente efectiva de los procesos mentales que efectúa el hombre, se debe a que ninguna de las formas de representación existentes permite realmente conceptualizar todo lo que involucra la tarea de pensar. Es por ello que vale la pena considerar la definición de alguna FRC nueva o, la definición de

mecanismos que permitan integrar adecuadamente las existentes, ya que, en conjunto las FRC ofrecen mejores alternativas.

Los distintos puntos en el área de Inteligencia Artificial que constituyen focos centrales para investigaciones futuras resultaron ser:

(1) Validación, mediante SBC arquetípicos, del tipo de resultados presentados en las tablas asociadas a cada una de las AG analizadas.

(2) Estudio detallado del potencial existente en las bases de conocimiento que resultan de la integración de diferentes formas de representación.

(3) Definición de una metodología para el desarrollo de SBC utilizando la clasificación del conocimiento que se expone en el informe y algunas de las técnicas existentes para la extracción del conocimiento de los expertos.

(4) Analizar la forma de la base de conocimiento requerida por un SBC definido con el objetivo de realizar más de una AG en forma simultánea.

(5) Definición de un SBC que permita a los diseñadores de este tipo de sistemas seleccionar las formas de representación que más conviene utilizar en la definición de la base de conocimientos del sistema que se está desarrollando. El proceso de selección puede estar basado en los criterios definidos en este trabajo.

AGRADECIMIENTOS

Queremos agradecer a los profesores Jorge Baralt y Alexander Zambrano por sus valiosos comentarios sobre los elementos de la investigación.

VII.-Bibliografia

BONNET A.,"L'intelligence Artificielle: Promeses et Réalités", InterÉditions 1984.

CALLERO M., WATERMAN D., KIPPS J.,"TATR: A Prototype Expert System for Tactical Air Targeting", Reporte R-3096-ARPA, Rand Corporation, Junio 1984.

CHANDRASEKARAN B.,"Generic Tasks in Knowledge-based Reasoning: High-level Building Blocks for Expert System Design", IEEE Expert, Octubre 1986.

CHANG KAI-HUNG, WEE W.,"A Knowledge-Based Planning System for Mechanical Assembly using Robots", IEEE Expert, Primavera 1988.

DUDA R., GASCHNIG J., HART P.,"Model Design in the PROSPECTOR Consultant System for Mineral Exploration", in Michie 1979.

FRANKLIN J., LACKEY C., KELLER K., LEVITT T., BUTEAU B.,"Expert System Technology for the Military: Selected Examples", Proceedings of the IEEE, vol. 26, Nº 10, Octubre 1988.

GEVARTER W.,"The Nature and Evaluation of Commercial Expert System Building Tools", COMPUTER, Mayo 1987.
PEREZ-MIÑANA E.,"Criterios de Selección para las Formas de Representacion del Conocimiento", Trabajo de Grado para optar al Título de Magister en Ciencias de la Computación en la Universidad Simón Bolívar, Octubre de 1990.
SLAGLE J., HAMBURGER H.,"An Expert System for a Resource Allocation Problem",Communications of the ACM, Nº 28, Septiembre 1985.
SOWA J.,"Information Processing in Mind and Machines", Ed. Addison-Wesley, 1984.
WATERMAN D.,"A Guide to Expert Systems" ,Addison-Wesley Publishing Company, 1984.

Un Manejador de Comunicaciones Minimal para Programas Síncronos sobre Redes de Transputers

Carlos Figueira

Departamento de Computación y
Tecnología de la Información
Universidad Simón Bolívar
Apdo. 89000 - Caracas 1080-A
Venezuela

Resumen

Este artículo presenta el diseño de un Manejador de Comunicaciones Minimal para aplicaciones, programadas en SIGNAL e implementadas en Redes de Transputers. SIGNAL es un lenguaje síncrono tipo flujo de datos para la especificación y programación de sistemas tiempo real. Luego de una etapa de validación, el programa SIGNAL es distribuido en la red de Transputers. El Manejador de Comunicaciones debe permitir las comunicaciones resultantes de dicha distribución.

Abstract

This article introduces the design of a minimal routing kernel, for applications, programmed in SIGNAL and implemented in a Transputer Network. SIGNAL is a synchronous data-flow language for real-time systems specifications and programming. After a validation stage, the SIGNAL program is distributed on the Transputer network. The routing kernel should permit the communications induced by the distribution.

1 Introducción

En los últimos años, hemos asistido a la aparición de un nuevo tipo de lenguajes, los lenguajes *síncronos* (ESTEREL[2], STATECHARTS[6], LUSTRE[3] y SIGNAL[12, 4]), diseñados para la especificación y programación de sistemas complejos de tiempo real, tales como el procesamiento de señales en aeronáutica, robótica, sistemas reactivos y reconocimiento de voz. La particularidad esencial de estos lenguajes es el manejo del tiempo según el principio que las acciones tienen una duración de tiempo lógico nula, liberando así al programador de los problemas

relacionados con la implementación, durante la etapa de diseño de los algoritmos. Por otro lado, este tipo de aplicaciones exige tiempos de respuesta muy rápidos, lo cual hace necesario el uso de multiprocesadores.

SIGNAL es un lenguaje tipo flujo de datos, desarrollado en el IRISA, Rennes, y forma parte de un proyecto de sistema de Diseño Asistido por Computador que engloba desde el desarrollo de los algoritmos de tiempo real hasta la implementación de estos en una arquitectura de multiprocesadores. El elemento básico manejado por el lenguaje se denomina *señal*: una señal puede ser considerada como una variable, de tipo definido, cuyo contenido es válido (y disponible) sólo en ciertos instantes lógicos, determinados por su *reloj*.

El compilador de SIGNAL verifica que el programa es correcto; en particular, que no contenga ciclos de dependencia, lo cual garantiza la ausencia de bloqueo.

En el contexto actual, la arquitectura prototipo consiste en una red de Transputers[7]. El Transputer es un microprocesador de 32 bits diseñado para ser utilizado como elemento de base de arquitecturas de multiprocesadores: es fácilmente interconectable gracias a sus 4 conectores seriales, posee una buena capacidad de cómputo entera y flotante (T800), y es programable en occam[9], que es un lenguaje de alto nivel.

Una vez seleccionada la topología de la arquitectura, el programa original, compilado, es particionado y cada partición es asignada a uno de los procesadores de la arquitectura. Es necesario entonces agregar el código de un *Manejador de Comunicaciones* en cada procesador, que se encargue de compartir los conectores seriales de los Transputers entre las diferentes señales que pasan por dichos conectores, y las señales que comunican al código de ese procesador con otros procesadores.

El código asignado a un procesador está compuesto por dos procesos:

- Un proceso *módulo*, que corresponde a aquella parte de la aplicación asignada al procesador.
- un proceso *sistema*, el cuál se encarga del ruteo, multiplexado, comunicación con el exterior y control del principio y fin de los instantes lógicos.

Estos dos procesos se implementan en *paralelo* en occam de la manera siguiente[1]:

```
...
PRI PAR
  sistema
  módulo
```

La ejecución de programas SIGNAL en la red de Transputers se rige por el principio de *sincronía fuerte*:

[1]Las ventajas de colocar en alta prioridad las tareas sistema ha sido reportada en [1, 13].

En cada instante lógico, cada procesador ejecuta las acciones que le corresponden a dicho instante, y luego espera a que todos los procesadores hayan terminado sus respectivas acciones para ese instante.

Esta sincronización entre los distintos procesadores se efectúa por intercambio de mensajes. Un árbol cobertor se define sobre la arquitectura. Así, cada nodo recibe de su padre la señal de comienzo de instante, la retransmite a sus hijos, y luego envía su padre la señal de fin una vez que reciba de sus hijos la señal de fin de instante, y que él mismo haya terminado. El procesador en la raíz del árbol concentra la decisión de comienzo y fin de instante.

El Manejador de Comunicaciones (llamado también MC en lo sucesivo) debe ser diseñado de manera tal que conserve las propiedades de ausencia de bloqueo del programa, ya verificadas por el compilador, pero minimizando la carga que él induce sobre el procesador. El artículo presenta cómo se alcanzan estos objetivos.

2 El Manejador de Comunicaciones

Los sistemas de enrutamiento utilizados en redes de computadores se clasifican en dos tipos, según el procesamiento que se aplica a los mensajes en los nodos intermedios:

- *store and forward*: los mensajes son recibidos, guardados en memoria y luego retransmitidos hacia su destino.
- *virtual cut-through*: los mensajes comienzan a ser retransmitidos antes de haber sido recibidos por entero.

Este último requiere de circuitería especializada. Una variante de éste método, denominada enrutamiento *wormhole*, ha sido implementada en VLSI para los hipercubos iPSC, de Intel, y Cosmic Cube, de Caltech. Las ventajas de disponer de circuitería especializada para el enrutamiento son evidentes; sin embargo, ésto sólo es posible para topologías regulares, tales como el hipercubo.

La interfaz de los conectores seriales del Transputer, la cuál funciona de manera concurrente a la unidad central del procesador, se presta bien para implementar un sistema de enrutamiento del primer tipo (store and forward), ya que esta interfaz funciona en modo DMA, es decir, en acceso directo a la memoria. La próxima generación de Transputers, el H1, incorporará circuitería para enrutamiento wormhole, por lo cual el análisis presentado aquí no se le aplica.

2.1 Principios del Manejador de Comunicaciones

El MC está constituido por un conjunto de procesos occam generados para cada procesador. El objetivo consiste en proveer los servicios de enrutamiento

sin introducir bloqueos, utilizando el mínimo de recursos. Los recursos en este caso son el espacio de memoria y tiempo del CPU.

Un bloqueo en una red de interconexión ocurre cuando ningún mensaje puede avanzar hacia su destino debido a que las colas de mensajes están llenas [10]. Diversos métodos para evitar bloqueos han sido propuestos en la literatura [5].

En nuestro contexto, disponemos de toda la información acerca de las comunicaciones en la red, de manera estática, es decir, durante la compilación. Esta consiste en:

- El tipo de los datos que serán transmitidos, ya que las señales comunicadas entre procesadores han sido declaradas en el programa;
- La ruta que siguen los datos.

Según Günther [5], para prevenir la aparición de bloqueos en sistemas en los cuales las comunicaciones son conocidas estáticamente, debe establecerse un *canal virtual* para cada comunicación.

Un canal virtual es un canal lógico que comparte canales físicos de la red, pero que tiene asignada una cola propia en cada nodo intermedio.

En ese contexto, las únicas causas posibles de bloqueo se reducen a aquellas inherentes a la aplicación[2]: si las colas no son vaciadas oportunamente, su saturación provocará el bloqueo de otras señales.

Para ver que esta situación no puede aparecer en nuestro caso, consideremos una cola de dimensión 1, situada en un procesador perteneciente a la ruta de una señal a. Puesto que las colas son únicas para cada señal, ningún otro mensaje puede llegar a esa cola; es decir, **no se introducen nuevas dependencias entre señales**.

Supongamos ahora que la cola está llena con un valor a_t de la señal "a", correspondiente al instante lógico t. Para que la cola se sature, un nuevo valor a_{t+1} de "a" debería ser producido por el procesador fuente. Pero, si a_t no ha podido alcanzar su procesador de destino, esto implica que el instante lógico t no ha terminado aún, puesto que el procesador de destino requiere ese valor en sus acciones del instante t. Esto contradice el principio de sincronía fuerte.

Dos consecuencias importantes para el enrutamiento derivadas del principio de sincronía fuerte son:

- Colas de capacidad para un sólo valor son suficientes para cada señal. Esto minimiza la cantidad de memoria utilizada por el manejador.
- No hay necesidad de implementar un control de saturación de las colas, lo cual reduce el uso del CPU por parte del MC.

[2]En la notación de Günther, éstas son denominadas *Propagación de Bloqueos Causados por el Usuario en la Red.*

El sistema de enrutamiento que implementaremos difiere ligeramente del presentado anteriormente, en que las colas de los nodos intermedios son desplazadas hacia los procesadores fuente y destino. Es obvio que las propiedades de ausencia de bloqueo mostradas aquí son conservadas por esta transformación.

2.2 Descripción del Manejador de Comunicaciones

El MC asociado a cada procesador está compuesto por varios procesos occam. La separación en varios procesos se debe a:

- garantizar la unicidad de las colas. Esto se logra al asignar un buffer de entrada y un buffer de salida para cada señal.
- explotar al máximo la concurrencia entre el CPU y las interfaces de los conectores.
- explotar la concurrencia entre los diferentes conectores de un Transputer.

Por cada conector serial de entrada:

- un proceso llamado *receptor*, el cual recibe los mensajes y los pasa al enrutador de ese conector (ver más adelante).
- un *enrutador de conector*. Este interpreta la información de enrutamiento del mensaje y decide a cual *buffer* debe enviarlo.

Por cada conector serial de salida: un proceso *multiplex*, el cuál recibe los mensajes de salida de los buffers interconectores y de los buffers de las señales de salida del procesador, y los envía por el conector serial.

Buffers inter-conectores, situados entre cada enrutador de conector de entrada y cada multiplex de salida, en los casos en que estos dos conectores pertenezcan a alguna de las rutas definidas en la aplicación.

Por cada señal de entrada del módulo de un procesador, un *buffer de entrada* que se encarga de recibir el mensaje del enrutador, extraer el valor que éste contiene y pasar dicho valor al módulo.

Por cada señal de salida del módulo, un *buffer de salida*, el cuál recibe el valor de la señal del módulo, le agrega la información de enrutamiento y envía el mensaje así compuesto al multiplex del conector de salida que corresponde a la ruta de dicha señal.

Note que todos los procesos, excepto los buffers de entrada y salida, pueden ser compartidos por otras señales. Estos buffers desempeñan el papel de las colas de los canales virtuales, con la diferencia que aquí han sido desplazadas hacia los procesadores fuente y destino.

La implementación de estos procesos es presentada en el anexo.

2.3 Medidas preliminares

Algunas medidas preliminares han sido tomadas sobre una plataforma inmos B003, con T414 corriendo a 15 MHz y conectores a 20 Mbits por segundo, con el objeto de obtener un orden de magnitud de la carga sobre el CPU ocasionada por el MC.

Las pruebas consistieron en el envío de un mensaje desde un nodo a sí mismo vía un nodo intermediario. Fueron hechas a la vez estimaciones usando el manual del Transputer [8], y medidas con el *timer* del Transputer.

El mensaje de prueba es un número de punto flotante de 64 bits, a los cuales se agrega el encabezado del mensaje (2 octetos) más un octeto que indica el largo del mensaje, lo cual dá un largo total de 11 octetos para el mensaje.

El tiempo tomado por ambas transmisiones, del nodo fuente al nodo intermediario, luego de vuelta al nodo fuente, sobre el conector fué de

$$28.8\mu s$$

El tiempo total en ambos Transputers fué de

$$753.5 ciclos(20MHz) \approx 37.67\mu s$$

Por último, el tiempo CPU sobre el nodo intermediario fué de:

$$366 cycles \approx 18\mu s$$

Las pruebas fueron hechas sin ninguna otra carga en los Transputers, y con sólo las transmisiones especificadas arriba. Otras medidas deben ser realizadas en situaciones realistas para poder evaluar de manera precisa el MC.

3 Conclusiones

Un Manejador de Comunicaciones adaptado a la ejecución de programas SIGNAL sobre una arquitectura a base de Transputers ha sido presentada. Su interés principal consiste en la metodología seguida para aprovechar las características de las aplicaciones para obtener un manejador de comunicaciones eficaz y seguro.

La eficiencia exigida para cumplir con las especificaciones temporales en las aplicaciones de tiempo real, justifica el empleo de diseños específicos en lugar de utilizar sistemas de enrutamiento generalizados, como es el caso de SYNDEX [11].

Por otro lado, el análisis de las posibles causas de bloqueo, facilitado por las propiedades estáticas de los programas SIGNAL y la verificación realizada por el compilador, nos guiaron hacia un diseño que logra evitarlos por completo.

Referencias

[1] P. Atkin. Performance maximisation. *inmos Technical Note 17*, 1987.

[2] G. Berry, P. Couronné, and G. Gonthier. Synchronous programming of reactive systems: an introduction to ESTEREL In K. Fuchi and M. Nivat, editors, *Programming of future generation computers*, pages 35–56. ICOT INRIA, North-Holland, 1988.

[3] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. LUSTRE: A declarative language for programming synchronous systems. In *14th ACM Symposium on Principles of Programming Languages*, pages 178–188, Munich, 1987.

[4] T. Gautier, P. Le Guernic, and Loïc Besnard. SIGNAL: a declarative language for synchronous programming of real-time systems. In G. Kahn, editor, *Functional programming languages and computer architecture*, pages 257–277. Lecture Notes in Computer Science, 274, Springer-Verlag, 1987.

[5] K. Günther. Prevention of deadlocks in packet-switched data transport systems. *IEEE Transactions on Communications*, Com-29(4):512–524, 1981.

[6] C. Huizing, R. Gerth, and W. P. de Roever. Modelling STATECHARTS behaviour in a fully abstract way. In M. Dauchet and M. Nivat, editors, *13th Colloquium on Trees in Algebra and Programming CAAP'88, Lecture Notes in Computer Science*, pages 271–294, Nancy France, March 1988. Springer Verlag. Volume 299.

[7] INMOS. *Transputer Reference Manual.* INMOS Ltd., 1000 Aztec West, Aldmondsbury, Bristol, England, 1986.

[8] INMOS. *The* TRANSPUTERS, *Reference Manual.* Bristol, U. K., inmos edition, 1986.

[9] INMOS. occam *2 Reference Manual.* Prentice Hall, 1988.

[10] L. Kleinrock. *Queueing Systems.* Wiley, 1976.

[11] C. Lavaranne and Y. Sorel. Syndex: un environnement de programmation pour multi-processeur de traitement du signal. Research report 113, INRIA FRANCE, 1989.

[12] P. Le Guernic, A. Benveniste, P. Bournai, and T. Gautier. SIGNAL: a data flow oriented language for real time signal processing. In C. I. Byrnes and A. Lindquist, editors, *Computational and combinatorial methods in systems theory*, pages 419–433. North-Holland, 1986.

[13] D. Pritchard, C. Askew, D. Carpenter, I. Glendinning, A. Hey, and D. Nicole. Practical parallelism using transputer arrays. In *Parle 87* Springer-Verlag, 1989. LNCS 258.

A Implementación

El formato de lo mensajes, así como el código occam del Manejador de Comunicaciones son presentados en este anexo.

A.1 Formato de los mensajes

Un mensaje es una secuencia de octetos, compuesto por:

1. tamaño (en octetos). en occam, es necesario especificar el número de octetos que posee el mensaje.
2. identificación de la ruta.
3. identificación de la señal.
4. el valor de la señal, el cuál es traducido en una secuencia de octetos a la transmisión y reconstruído a la recepción.

Los mensajes son formateados en los buffers de salida. El valor es extraído en los enrutadores de conector.

A.2 El receptor

Su función primordial es separar las tareas de recepción por el conector, que es una operación lenta, de la tarea de enrutamiento. Su estructura es:

```
[maxsize]BYTE message:
BYTE size.mess:
WHILE TRUE
  SEQ
    Linki ? size.mess::message
    Ri.Routi ! link.mess::[message FROM 0 FOR INT size.mess]
```

- Linki significa el conector "i"
- Linki.Routi conceta el receptor de este conector a su enrutador de conector Ri.

A.3 Enrutador de conector

Este proceso contiene la información para el enrutamiento de los mensajes que llegan por ese conector. La información es representada por dos funciones: *gen.route* and *loc.route*. La primera devuelve, para la ruta de un mensaje, el buffer interconector al cuál debe ser enviado el mensaje. En el caso que este procesador sea el destino, la segunda función provee el buffer de entrada, de acuerdo a la identidad de la señal contenida en el mensaje.

El proceso enrutador Routi tiene las siguientes conexiones:

- a los buffers interconectores vía un arreglo de canales "Routi.BiLi[dest]", donde el índice "dest" selecciona el buffer interconector.
- a los buffers de entrada vía un arreglo de canales "Routi.IBi[loc.dest]"; el índice "loc.dest" selecciona el buffer de entrada al cual el valor es enviado.

El código occam es:

```
[maxsize]BYTE message:
BYTE size.mess,dest,loc.dest:
WHILE TRUE
  SEQ
    Ri.Routi? size.mess::message
    dest := gen.route(message[0])
    IF
      dest <> 0
        Routi.BiLi[dest] ! size.mess::[message FROM 0 FOR INT size.mess]
      dest = 0
        SEQ
          loc.dest := loc.route(message[1])
          Routi.IBi[loc.dest] ! (size.mess-2)::[message FROM 0 FOR INT size.mess-2]
```

A.4 Buffers Interconectores

Estos conectan al enrutador de un conector de entrada al multiplex de un conector de salida, como se especifica a continuación:

- al enrutador de conector a través del canal "Routi.BiLi[ind.in]", donde "id.in" es una constante que identifica el elemento del arreglo de canales que lo conecta al enrutador.
- al multiplex de salida a través del canal "B.Mux[ind.out]", donde "ind.out" es una constante que indica el canal que lo conecta al multiplex.

```
[maxsize]BYTE message:
BYTE size.mess:
VAL ind.in IS ...:
VAL ind.out IS ...:
WHILE TRUE
  SEQ
    Routi.BiLi[ind.in] ? size.mess::message
    B.Muxj[ind.out] ! size.mess::[message FROM 0 FOR INT size.mess]
```

A.5 Buffer de entrada

El código occam es:

```
TYPE a.v :
VAL dest IS ...:
WHILE TRUE
  SEQ
    Routi.IBi[dest] ? a.v
    a ! a.v
```

donde

- "Routi.IBi[dest]" conecta el buffer de entrada con el enrutador de conector; la constante "dest" indica el canal del arreglo que lo conecta al enrutador de conector.
- "a" es el canal que conecta el buffer al módulo.

A.6 Buffer de salida

```
TYPE x.v :
VAL l IS ...:
VAL BYTE route IS ...:
VAL BYTE ident IS ...:
VAL BYTE size.sig IS ...:
VAL BYTE size.mes IS ...:
WHILE TRUE
  SEQ
    x ? x.v
    B.Muxi[l] ! size.sig;route;ident; signal.v
```

- "x" es el canal que conecta el módulo al buffer de salida.
- "B.Mux[l]" conceta el buffer al multiplex de salida del conector "i".
- "route" e "ident" son los identificadores de la ruta y de la señal, respectivamente.
- "size.sig" es la cantidad de octetos según el tipo de la señal.
- "size.mess" es el número de octetos del mensaje entero.

A.7 Multiplex de salida

Se implementa de la siguiente manera:

```
VAL INT n IS :
[max.size]BYTEmessage :
BYTE size.mes :
WHILE TRUE
  SEQ
    ALT j=0 FOR n
      B.Mux[j] ? size.mes::message
        SKIP
    linkj ! size.mess::[message FROM 0 FOR INT size.mess]
```

- "n" es el número de canales que conecta al multiplex con los buffers de salida y los buffers interconectores.
- "max.size" es el largo máximo (en octetos) de los mensajes que atraviesan este conector.
- "linkj" es el conector de salida.

S.O.O.M.
Un Modelo Semántico para Bases de Datos

Ing. Augusto Giancola, Ing. Francisco Marcano, Ing. Leonid Tineo
Ing. Ma. Esther Vidal, M.Sc. Enzo Chiariotti

Universidad Simón Bolivar, Caracas, Venezuela

Es creciente el interés en la incorporación a los manejadores de bases de datos de herramientas de modelación que expresen no sólo la estructura de los datos, sino también su intepretación y su papel en el sistema en que se encuentran. Como consecuencia de este interés encontramos han venido siendo explorados múltiples caminos para la modelación de datos.

Se presenta una revisión de los principales conceptos asociados a la modelación de datos y de diferentes paradigmas y proposiciones en el área, para, sobre la base de tales conceptos y proposiciones presentar la definición de un modelo de datos semántico, el modelo S.O.OM., el cual constiuído por un conjunto de abstracciones que favorecen la expresión de manera sencilla y natural de la estructura y significado de los datos los sistemas reales que se modelen.

INTRODUCCION

Los manejadores de bases de datos mayormente difundidos y conocidos, dejan de mano del programador el conocimiento y la preservación del significado de los datos que se manipulan; esto como consecuencia de la carencia en tales manejadores -o en los modelos que soportan- de herramientas de modelación suficientes para expresar no sólo la estructura de los datos, sino también su intepretación.

Se presenta en este trabajo la definición de un Modelo de Datos semántico, constituído por un conjunto de conceptos y abstracciones, dirigidas a constituir para el modelador de datos herramientas manejables y expresivas a los fines de recoger tanto estructura como significado de los datos de un sistema.

La definición del Modelo S.O.O.M. que aquí se presenta forma parte del proyecto de Manejador de Bases de Datos que se adelanta en la Universidad Simón Bolivar, y como parte del cual los autores han realizado también una definición formal del modelo y la definicion e implementación de un prototipo de Lenguaje de Definición de Datos para el modelo.

La presentación del modelo S.O.O.M. se realiza en cuatro capitulos:

El primero expone un conjunto de conceptos, que constituyen el marco teórico-conceptual relativo a la modelación de datos, con especial atención sobre sus objetivos, tanto de carácter general, como aquellos asociados a la implementación de un modelo.

En el segundo capítulo se presentan los resultados de un análisis comparativo del universo de modelos de datos estudiados, realizado sobre la base de los aspectos que, en el marco conceptual del capítulo precedente, se consideran resaltantes en la evaluación de un modelo.

Partiendo la revisión de conceptos del segundo capítulo, el tercero, se hace una selección y justificación de aquellos que serán soportados por el modelo S.O.O.M, definido en el cuarto capítulo, que también incluye la representación gráfica de tales abstracciones y un ejemplo de modelación.

1. MARCO CONCEPTUAL

Se presenta una revisión de lo que se considera el conjunto básico de conceptos sobre los cuales, en los capítulos posteriores, se basará la presentación, análisis y evaluación de diferentes paradigmas de modelación de datos, y que orientarán la definición del modelo S.O.O.M.

1.1. Percepción y Expresión

La interacción del ser humano con su ambiente se verifica a través de dos procesos fundamentales, la **percepción** y la **expresión**. Por el primero, el hombre captura información de su ambiente, alimentando su visión particular del mismo; por el segundo, es capaz de transmitir a este mismo ambiente, la visión particular que ha elaborado del mundo real.

En el primer proceso, el hombre recibe de su entorno sensaciones, las cuales, enriquecidas con conceptos que ya ha elaborado, generan nuevos conceptos y, posiblemente, modifica los anteriores; a este proceso asociado a la percepción, lo denominamos **conceptualización**, y es a partir de él que cada hombre construye su visión del entorno.

Cuando el intercambio se verifica desde el ser humano hacia su entorno, es decir, cuando aquel desea expresar la visión del mundo real que ha construido, encontramos otro proceso de elaboración sobre el conjunto de conceptos que forma esa visión; a esta elaboración la denominaremos **modelación**; se trata de una reelaboración que represente de la manera más fiel y exacta posible ese conjunto de conceptos.

En el proceso de modelación, es necesaria la habilidad de ocultar los detalles y particularidades de los múltiples elementos que constituyen nuestro conocimiento, concentrándose en aquellos aspectos comunes de conjuntos de ellos, a los fines de reducir la complejidad y aumentar la significación de lo expresado. A esta habilidad la llamamos **abstracción**. Al conjunto de abstracciones y conceptos involucrados en la modelación,

constituye lo que denominaremos un **modelo**.

Para esa interacción del ser humano con su entorno -el cual incluye otros seres humanos-, son múltiples los modelos que ha elaborado, cada uno influenciado por diferentes y específicas necesidades de expresión: modelos más sofisticados permiten expresar fenómenos complejos. La necesidad de comunicación agrega la necesidad de encontrar modelos comunes para la expresión de ciertas ideas (lenguaje natural, formalismos matemáticos, etc).

1.2. Los modelos de datos

Como se mencionó, el hombre trata de describir los fenómenos que percibe de su entorno, y que constituyen la base del conocimiento de éste. Cuando hablamos de **datos** hablamos de estas descripciones que el ser humano hace de cada uno de los hechos (fenómenos) que percibe, y de las posibles relaciones que entre ellos haya percibido -todos estos hechos concretos-, o, inclusive, inferido.

El problema de la modelación de datos es el mismo que se ha descrito cuando hablamos de la expresión que hace el ser humano de su visión de mundo real -o de parte de éste-. Prácticamente cualquier forma de expresión conocida del ser humano constituye un caso de modelación de datos, cada una sobre su propio **modelo**, cada uno tratando de capturar la esencia del significado de los datos expresados.

Dos objetivos se reconocen claramente en la construcción y utilización de cualquier modelo de datos: expresividad y economía conceptual. El primero está relacionado con la necesidad de transmitir la esencia de lo representado, el segundo, apunta hacia la manejabilidad, del conjunto de abstracciones que constituyan al modelo. Las vías hacia cada una de estas metas no siempre coinciden, ni convergen, por lo que la modelación implicará siempre el compromiso de reducir el conjunto de conceptos (y abstracciones) utilizados, preservando el poder de representar la realidad de manera natural.

Consideraremos un **modelo de datos** como una herramienta intelectual que permite una interpretación del mundo real, suficientemente abstracta para favorecer su manejabilidad y, al mismo tiempo, suficientemente poderosa para ofrecer entendimiento sobre cómo están relacionados los datos sobre el mundo real. [Tsichritzis 82]

1.3. Modelos de Datos y Ambientes Automatizados de Manipulación de Datos

En la medida en que la representación de la realidad -o de una fracción de ella- se hace compleja, y

con esto, mayor el volumen de datos y relaciones que entre estos deben ser consideradas, crece la necesidad, y el interés, por la manipulación automatizada de estos datos y relaciones. Es este el caso del manejo de datos corporativos en una empresa, o del cálculo en un ambiente de investigación —para el que se desarrollan los primeros sistemas de procesamiento de datos—, pero también lo es el de la representación del conocimiento, los ensayos en inteligencia artificial -que, de manera particular, han enriquecido notablemente el área de la modelación de datos-.

De la necesidad de automatizar el manejo de los datos usados en la representación de la realidad, se desprende un tercer objetivo general a perseguir en la selección de las abstracciones y conceptos que constituyan el modelo: el de considerar la factibilidad de su implementación eficaz y eficiente en el ambiente automatizado que soportará esa interpretación de la realidad. Como en el caso de los enunciados anteriormente, este objetivo implica transitar un camino que no coincide con los de los otros, por lo que ha de establecerse un nuevo compromiso: cuáles abstracciones deben ser eliminadas del modelo por su dificultad de manejo automatizado, cuáles se siguen incluyendo, aunque no sean soportadas por una implementación -debiendo ser manejadas por el "usuario" del modelo-, cuáles deben ser preservadas -o agregadas- para facilitar la implementación.

Es sobre estos modelos de datos, orientados a la representación de la realidad en organizaciones complejas en vista del manejo automatizado de la información, que se desarrolla nuestro trabajo; y sobre los aspectos mencionados en el párrafo anterior y la satisfacción de los objetivos generales, que se conduce en el capítulo siguiente una revisión de los diferentes paradigmas de modelación de datos.

2. PARADIGMAS PARA LA MODELACION DE DATOS

Se presenta a continuación el resultado de la revisión de las principales orientaciones seguidas en la modelación de datos -como herramienta de diseño y en ambientes de manejo automatizado- hasta el día de hoy, señalando de qué manera y en qué medida, se han aproximado a la satisfacción de los objetivos generales enunciados en el capítulo anterior. Antecediendo a esta revisión, se indican los criterios para la agrupación de los modelos que se utiliza. Finalmente, se presentan dos cuadros comparativos, en los que se ubican los diferentes paradigmas, según elementos de evaluación que se describirán.

2.1. Criterios de Agrupación, Criterios para la Evaluación.

La revisión de paradigmas de modelación cuyos resultados aquí se refieren, se realizó agrupándolos

en dos categorías [Tsitchritzis 82]; el primero, corresponde a los más ampliamente utilizados y estudiados, que llamaremos **tradicionales**; el segundo incluye aquéllos orientados a la representación del significado de los entes y relaciones modelados, a los que, adoptando una denominación muy general, llamaremos **semánticos**. El análisis se realizó sobre la base de los objetivos de expresividad, economía conceptual -manejabilidad de conceptos- e implementabilidad, ya descritos, sin atender de manera particular a detalles de implementación, definición u operación de cada modelo.

2.2. Modelos Tradicionales

Entre los modelos tradicionales incluímos a **jerárquico, de redes** y **relacional**.

Estos modelos, en particular los jerárquicos y de redes, constituyen los primeros estadios en el reemplazo de los Sistemas de Archivos, utilizados hasta ese entonces en los ambientes de Procesamiento de Datos, por una representación de los datos que permitiera aprovechar las características particulares de estos, dándoles así significado, de manera independiente de las necesidades y aplicaciones a satisfacer -y de los cambios en éstas-.

Siendo este el marco en que surgen y se incorporan al mercado los modelos tradicionales, se explica que hayan seguido la vía de la implementabilidad, y que el conjunto de abstracciones soportadas fuera lo más reducido y manejable posible. La expresión del "significado" de los datos, aunque en cada modelo con diferente nivel, se encuentra orientada claramente a las operaciones (condiciones sobre ellas, especificación de las mismas), permitidas sobre los datos. (restricciones de inserción y pemanencia en los de redes, control por aplicaciones en el relacional).

Característica común de los tres modelos es el uso de tablas para la representación de los "entes" del sistema a representar, estructura muy familiar para los usuarios en el ambiente de procesamiento de datos. Sin embargo, cada modelo, posee características que los diferencia en la satisfacción de los objetivos generales de la modelación.

Modelos de Redes:

Los modelos de redes -entre los que se puede considerar al jerárquico como un caso particular- utilizan al grafo (con algunas restricciones), como estructura para la modelación. De esta manera, estos modelos permiten **expresar** relaciones entre entes del sistema modelado, sobre las cuales se pueden definir las reglas de inserción y permanencia ya mencionadas. De manera que cierta expresividad es ofrecida por el

modelo. No obstante, la orientación a la implementación de estos modelos, induce limitaciones sobre la generalidad de las relaciones: en el caso de redes, debe preservarse "propiedad exclusiva" de un ente sobre sus miembros, en el jerárquico, las relaciones son de carácter estrictamente jerárquico. Otras relaciones más generales (muchos a muchos, p.ej.), pueden representarse, pero con la introducción de entes artificiales -no forman parte de la realidad modelada-.

Modelo Relacional

En este caso, la unica estructura es la tabla, asociada a la única abstracción soportada: la agregación, vista como relación matemática. Esto reduce altamente la expresividad del modelo, pues, si bien permite representar de manera natural a cada ente del sistema, no hace lo mismo con las relaciones entre éstos, las cuales son representables también como tablas, pero comprometiendo al modelador a "recordar" (a través de la documentación) y preservar (por programas de aplicación) el significado de cada tabla. La menor expresividad va nuevamente en favor de la implementabilidad del modelo, pero de una manera distinta a la expuesta para los modelos de redes: las estructuras (red, jerarquía) permitidas inducen claramente una implementación (listas, listas doblemente enlazadas), en el caso relacional, las tablas constituyen un nivel de abstracción superior, el cual puede ser soportado a través de diferentes implementaciones.

Para la falta de expresividad que muestra el modelo relacional, el más utilizado actualmente, se han encontrado dos tipos de soluciones. La primera consiste en el uso de modelos más expresivos (E-R o E-R-E, [Chen 76], [Teorey 86]), en las tareas de diseño y documentación de la base de datos. La segunda, más formal, ha consistido en agregar al modelo algunos de los aspectos semánticos de los que carece (representación explícita de relaciones, preservación de integridad referencial); el intento más notable en este sentido lo constituye el modelo RM/T [Codd 79].

2.3. Modelos Semánticos.

Los modelos que se agrupan como semánticos se caracterizan por proveer conjuntos de abstracciones más completos, para la representación del significado de las diferentes relaciones que pueden encontrarse en el sistema que se trata de modelar, así como la posibilidad de expresar de manera explícita las diferentes restricciones que deban asociarse a cada ente o relación del sistema real.

Estos modelos se pueden clasificar en cuatro grupos [Tsitchritzis 82]: Modelos Entidad-Relación, Modelos Binarios Semánticos, Modelos de Redes Semánticas y Modelos Infológicos. Todos estos modelos

persiguen la expresividad e involucran implementaciones más complejas. Los modelos presentan en orden creciente en cuanto a expresividad y, como es natural, de dificultades en su implementación.

2.3.1. Modelos Entidad-Relación

Esto modelos ([Chen 76], [Tsitchritzis 82], [Peckham 88]), basados en tablas y grafos, han sido ampliamente aceptados y utilizados en el ambiente de Bases de Datos, especialmente como herramienta para Diseño y Documentación de Sistemas, -en particular en ambientes que utilizan manejadores relacionales-.

Las características que permiten incluirlos entre los modelos semánticos son las siguientes:

- Diferenciación **entidades** y **relaciones**. Las relaciones son genéricas, y pueden agrupar a una o más entidades (n-arias), pudiendo establecerse para cada entidad participante en ella, su grado máximo de participación, soportando el carácter de relación de una determinada tabla.
- Permite establecer **restricciones de dominio** (conjuntos válidos de valores), para los diferentes parámetros que constituyen a una entidad o una relación, las cuales también deben ser preservadas por un manejador que soporte el modelo.
- Una adición semántica importante, la constituye el soporte específico de relaciones con características especiales, pero frecuentes en los sistemas que se trata de modelar [Peckham 88]. Así, en el caso del Modelo Entidad-Relación se tienen relaciones de Identidad y de Existencia, cuyas especiales características son preservadas por el modelo; en el E-R-Extendido [Teorey 86], estos casos especiales no se incluyen, mientras sí se modelan las abstracciones de Clasificación y Tipificación que en el capítulo 4 serán expuestas, y que el modelo provee como jerarquías de Generalización y de Subconjunto.

3.3.2. Modelos Binarios Semánticos

Estos modelos, son los que muestran mayor economía conceptual entre los aquí agrupados; proveen como única estructura el **grafo**, sobre el cual se soportan los conceptos de **categoría** y **relación binaria**, conceptos sobre los cuales pueden construirse abstracciones más complejas (relaciones n-arias, agregación, restricciones de dominio, etc.). Esta limitación a categorías y relaciones binarias, aunque no impide, como se dijo, la representación de realidades más complejas, obliga al modelador a mantener el control de la semántica de estas construcciones -algo similar a la tarea del usuario del modelo relacional-. Sin embargo, el modelo ofrece dos herramientas valiosas en la reducción de esta tarea de control a nivel de programa de aplicación: a) la definición de restricciones de cardinalidad; b) la especificación -asociada a las relaciones y

categorías- de programas, con los que, a tiempo de modelación, se preservan las características especiales de cada relación o categoría, en el sistema modelado.

3.3.3. Modelos de Redes Semánticas

Estos modelos surgen como producto de la investigación en las áreas de Inteligencia Artificial y Representación del Conocimiento, y en consecuencia , se destacan por perseguir la representación de un subconjunto, suficientemente poderoso, de las abstracciones que utiliza el ser humano, en la expresión de su visión de la realidad. En particular, el área de Bases de Datos se ha enriquecido de este trabajo, tomando aquellas abstracciones que permitan aproximar el trabajo de modelación de un sistema de datos, a la visión que el modelador pueda hacerse de él. Como las características comunes más resaltantes de los modelos de redes semánticas se pueden destacar [Tsichritzis 82]:

- Las estructuras manejadas por el modelos son grafos (arcos y nodos), permitiéndose en un esquema la coexistencia de instancias (ocurrencias) y tipos (clases).
- Diferenciación entre tipo (clases) y *tokens* (instancias). Es el primero de los modelos presentados que introduce los términos de instanciación y generalización.
- Distancia Semantica: Nodos cercanos en el grafo se consideran semánticamente cercanos (los arcos de irrelevancia permiten "alejar" nodos, que en el grafo aparezcan cercanos).
- Partición: permite restringir el ámbito de una operación, a un subconjunto (de clases o de objetos) del esquema.
- Jerarquía de tipos: introduciendo las abstracciones de **Generalización** (IS_A) **y Agregación** (PART_OF), estableciéndose explícitamente reglas de herencia de las propiedades (relaciones, atributos, restricciones) de una clase a otra.

Los nodos pueden representar: CONCEPTOS: instancias de tipos que se encuentran en la BD; EVENTOS: acciones entre conceptos; CARACTERISTICAS de un concepto; VALORES: de una característica. Por su parte, los arcos se distinguen como: ASERCION (concepto-concepto); INSTANCIACION (concepto-tipo); RELACION BINARIA (tipo-tipo), la cual puede ser de jerarquía (IS_A o PART_OF), o genérica.

Se pueden especificar restricciones, como hechos (facts) acerca de *tokens* o clases, a través del cálculo de predicados; mientras otras más complejas, pueden especificarse a través de programas, asociados a los diferentes entes y operaciones, que forman parte del modelo como una metaclase predefinida.

Es importante mencionar que el crecimiento del conjunto de abstracciones que soporta el modelo, no afecta seriamente la economía conceptual, pues se trata de abstracciones que el ser humano maneja, y encuentra en su entorno, con frecuencia.

3.3.4. Modelos Infológicos

Estos modelos están orientados a usuarios terminales, tanto a tiempo de diseño como de operación ([Sudgren 74], [Tsichritzis 82]). A tiempo de diseño persiguen: a) facilitar la definición precisa de **qué** información debe contener la B.D., sin anticipar decisiones de **cómo** representarla en alguna estructura; y b) facilitar a expertos la transformación sistemática de la estructura de cierta información en una estructura eficiente de almacenamiento de datos. A tiempo de operación persiguen: a) definir una manera **no procedimental** de recuperar la información que requiere el usuario, sin tener que referirse a la estructura datalógica de la B.D.; y b) que el sistema manejador de Base de Datos **interprete** el requerimiento de información y seleccione una estrategia de procesamiento eficiente de manera dinámica.

El enfoque infológico muestra la orientación que siguen los modelos de Base de Datos, en cuanto a aumentar la expresividad del modelo, reduciendo la labor de traducción de la realidad que debe realizar el modelador, y la de interpretación que debe realizar el usuario. La entrada de un manejador que implemente el Modelo Infológico, debería aproximarse al lenguaje natural, la salida, debería aproximarse a lo que el usuario quiere saber. Esto representa el extremo del continuo de expresividad-manejabilidad sobre el que mencionamos se pueden ubicar los modelos de datos, pero también implica alcanzar el extremo contrario (el inferior) en el continuo de la implementabilidad -la otra medida en la evaluación de los modelos que se esta haciendo-: implementar un modelo bajo la perspectiva infológica implica interpretación el lenguaje natural e interpretacion y manejo del conocimiento, ambos, objetivos no alcanzados aun, y perseguidos por diferentes ramas de la computación.

2.4. El Paradigma Orientado por Objetos y las Bases de Datos

Desde Simula [Birtwistle 73], son muchos los aportes que se han dado en el campo de los lenguajes de programación orientados por objetos. [Nierstrusz 89] muestra un conjunto de conceptos relacionados con las construcciones y los mecanismos que deberían ser provistos por lenguajes que se desenvuelven sobre dicho paradigma.

Como consecuencia de este desarrollo, surge la idea de sistemas orientados por objetos, los cuales

proveen el soporte de tiempo de ejecución y el ambiente se desarrollo de aplicaciones orientadas por objetos. En relación al soporte de tiempo de ejecución, estos sistemas hacen referencia al manejo de objetos que engloba mecanismos tales como nombramiento de objetos, persistencia, concurrencia, distribución, seguridad y control sobre versiones de tipos y otros, como por ejemplo, recolección de basura. Por otra parte, proveen mecanismos poderosos para el desarrollo de aplicaciones, haciendo esta tarea más fácil, mediando el concepto de reusabilidad y herramientos de diseño y manejo de evolución de objetos.

Las manipulaciones anteriormente destacadas, pueden ser enfocadas desde el punto de vista de la tecnología de las Bases de Datos. Una Base de Datos Orientada por Objetos, es un sistema que provee soporte para objetos persistentes (transacciones, *queries*, etc.). Esto no es más que la extensión de los criterios tomados en el campo de lenguajes de programación orientados por objetos. Sin embargo, la incorporación de las ideas de este paradigma en el área de Bases de Datos, enfrenta algunos problemas por ser solucionados:

- inexistencia de un modelo de datos estándar;
- inexistencia de un lenguaje de manipulación de datos que sea equivalente al algebra relacional.
- inexistencia de lineamientos para el diseño de Bases de Datos Orientadas por Objetos.

los cuales deberán ser resueltos a fin de preservar los principios fundamentales del paradigma.

2.5. Comparación

En la revisión se ha destacado la manera en que los diferentes paradigmas de modelación se han aproximado a los tres objetivos generales enunciados. Indicamos, el marco histórico de la aparición de los modelos tradicionales, para mostrar la influencia que éste ejerció; mencionamos la orientación semántica de los modelos que agrupamos como tales, los cuales se presentaron en orden creciente de expresividad.

Para complementar esta visión, y ofrecer una ubicación comparativa de los modelos expuestos, se muestran a continuación dos gráficos, en los que los modelos son colocados sobre sendos continuos, asociados con dos criterios que consideramos determinantes en la evaluación de los modelos. El Gráfico 1 ubica a los modelos, en función del esfuerzo de traducción que debe hacer el modelador para representar la situación que está modelando. Los extremos de este continuo, hombre y máquina, representan, la orientación del modelo, esto es, hacia la visión del modelador, o hacia la implementación del sistema modelado.

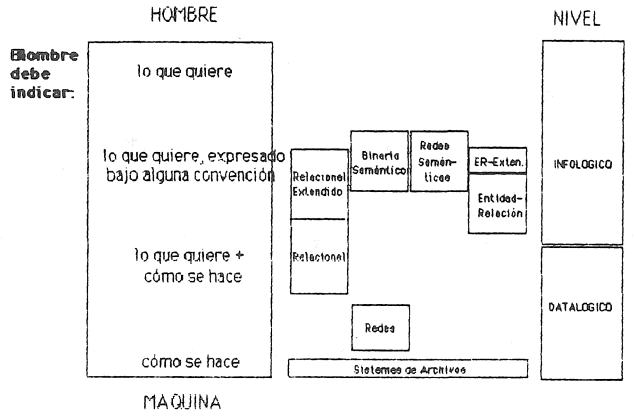


Gráfico 1

En el segundo (Gráfico 2) se ubican los modelos dentro de la perspectiva de la expresividad, esto es, de la manera en que permite representar la realidad, tal y como es.

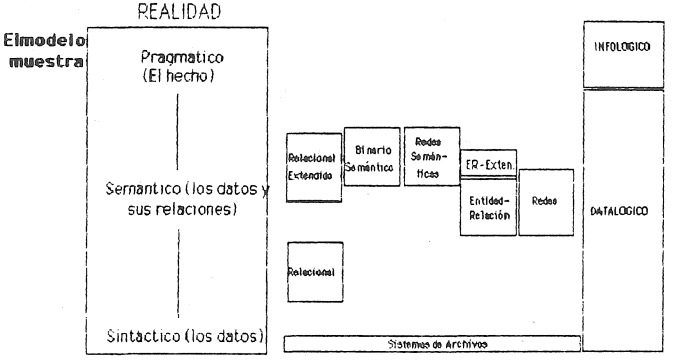


Gráfico 2

3. SELECCION Y DEFINICION DE UN MODELO

La formulación de un Modelo de Datos, implica la selección de características de uno o más paradigmas, que sirvan de base al conjunto de conceptos y abstracciones a ser soportadas por éste. La revisión de modelos cuyos resultados se reportaron en el capítulo anterior, constituyó un primer paso en esta tarea, en cuanto permitió conocer las orientaciones seguidas por cada modelo. El siguiente paso lo constituye la revision de los conceptos y criterios de trabajo, que el estudio de la modelacion de datos ha encontrado como fundamentales, a los fines de alcanzar expresividad en la modelación. Con estos conceptos se completa el

marco de análisis previo a la definición del modelo S.O.O.M.

3.1. Un Modelo Semántico

La orientación que sigue la modelación de datos es la de la expresividad, en cuanto a la representación del sistema real, en base a conceptos y abstracciones que lo reflejen fielmente. La definición de un modelo de datos, debe seguir esta orientación -precisamente la que sigue los modelos que llamamos **semánticos**- y es la que consideramos debía seguir el modelo cuya definición se reporta

Los aportes que los modelos semánticos ofrecen al campo de la modelación de datos (cada uno en diferente medida), y que justifican la selección de este paradigma para la definición de nuestro modelo, se pueden agrupar bajo cuatro criterios generales [Peckham 88]: Economia de Expresión, Preservación de la Integridad, Flexibilidad en la Modelación y Eficiencia en la Modelación.

La economia de expresión es aportada por los modelos semánticos al permitir extraer y representar un amplio rango de la información del sistema que modela, mediante un conjunto limitado de abstracciones, cuya naturaleza es similar a la de aquéllas usadas por el ser humano en su comprensión del ambiente, facilitando el proceso de modelación. La introducción de tales abstracciones, favorece la preservación de la integridad de los datos, al proveer mecanismos, implícitos en cada abstracción o definibles por el modelador, para el soporte de las restricciones asociadas a cada ente de un esquema.

La flexibilidad en la modelación, se refiere a la posibilidad que ofrecen los modelos semánticos de representar las situaciones reales, utilizando los conceptos y abstracciones que mejor se ajusten al papel del sistema modelado, flexibilidad que no proveen los modelos "tradicionales", cuyas posibilidades de representación se circunscriben a las estructuras que proveen (tablas, tablas y redes o jerarquías).

Por último, la eficiencia en la modelación, se alcanza en la medida en que el modelador debe ocuparse menos de los detalles de implementación. Los modelos semánticos proveen, asociados a las abstracciones que soportan, mecanismos predefinidos para la preservación de su significado, evitando al modelador mantener este control cada vez que un mismo de tipo de relación o abstracción sea utilizada.

3.2. Revisión de Diferentes Modelos Semánticos:

Revisadas las ventajas generales del uso de la modelación semántica, la definición del modelo requiere la revisión de diferentes proposiciones que se encuentran dentro del marco de los Modelos de Datos

Semánticos, con el objetivo de conocer las facilidades que ofrecen, los aspectos en que coinciden y aquéllos en que difieren.

Los modelos revisados fueron : Entidad-Relación [Chen 76], RM/T [Codd 79], TAXIS [Nixon 87], SDM [Hammer 81], Funcional [Shipman 81] , SAM* [Su 83], Event [King 84] y SHM+ [Brodie 84] -de los que se obtuvo información indirecta [Tsitchritzis & Lochosky 82][Peckham & Maryanski 88]-; e IFO. De esta revisión, reportamos los aspectos en que las diferentes propuestas coinciden, y aquellos en los que se encuentran diferentes enfoques [Peckham 88]

Los diferentes modelos coinciden en:

- Proveer relaciones entre objetos de datos, que soporten la manera en que el usuario percibe el sistema real.
- Para tales relaciones deben ser provistos mecanismos para especificar estados, transiciones y respuestas válidas para la B.D.

mientras existen diferentes enfoques respecto a:

- La generalidad del modelo: ¿debe ser independiente de la aplicación u orientado hacia algún ámbito en particular?
- Debe proveerse en un alto nivel la preservación de la semántica de las relaciones soportadas, o dejar al modelador, la posibilidad de controlar las restricciones asociadas a ellas.
- Deben proveerse relaciones y abstracciones complejas (agregación, tipificación, colecciones, etc.) o, por el contrario, primitivas y generales (modelos binarios).

3.3. Conceptos y Abstracciones soportadas por los diferentes Modelos Semánticos.

La construcción de modelos semánticos implica la evaluación de diferentes conceptos y abstracciones, a los fines de establecer cuales deben ser soportadas por los modelos, cuáles no, y aquéllos que se pueden construir a partir de otras de menor nivel. Las abstracciones se presentan en la sección 3.3.1, mientras que en la sección 3.3.2, se harán algunas consideraciones sobre un concepto central en la modelación semántica: la definición por intensión -y no por extensión- como una manera de aumentar el poder de expresión y la flexibilidad de un modelo.

3.3.1. Abstracciones

Se presenta un conjunto representativo de las abstracciones utilizadas por el hombre en la expresión

de su visión de su entorno, las cuales son soportadas por diferentes modelos semánticos.

3.3.1.1. Agregación.

En la realidad, muchos objetos pueden ser vistos como constituidos por partes diferenciadas, cada una con significado propio. *La Agregación es la abstracción que permite definir un concepto compuesto a partir de objetos más simples (partes)* [Smith 77a,b].

Ejemplo 3.3.1.1: En un archivo, cada registro es la **agregación** de un grupo de campos:

Código	Nombre	Dpto.	Salario

3.3.1.2. Generalización.

En ciertas ocasiones, es necesario considerar un conjunto de objetos como similares, aún cuando puedan presentar diferencias estructurales. *Generalización es la abstracción que permite crear un* ***concepto genérico*** *a partir de conceptos con propiedades semánticas comunes.* [Smith 77b].

> **Ejemplo 4.3.1.2**: Barcos, carros y aviones se pueden abstraer en el concepto genérico como vehículos de transporte, a pesar de sus diferencias estructurales.

4.3.1.3. Clasificación

Agrupar objetos de un mismo tipo con propiedades particulares comunes, en clases disjuntas (tales que su unión represente el conjunto inicial de objetos en cuestión) es el mecanismo provisto por la **clasificación.**

> **Ejemplo 4.3.1.3**: En una fábrica de baldosas, estas se **clasifican** según el material que las constituyen, en baldosas de alta, media y baja calidad.

4.3.1.4. Tipificación

Un mecanismo similar a la clasificación, donde las clases resultantes no son necesariamente disjuntas y el criterio de pertenencia para cada clase se puede formar a partir de distintas propiedades, constituye la **tipificación.**

> **Ejemplo 4.3.1.4.**: En una librería se quiere tener una Base de Datos sobre Temas de Libros. Podrían tipificarse los libros como Libros de Inteligencia Artificial, Libros referentes a Computación Gráfica y Libros con temas de B.D. Es claro que un libro puede hablar de todos los temas, o sobre ninguno. Por ejemplo, un libro que trate sobre el Modelo Semántico puede estar entre los libros sobre Inteligencia Artificial y entre aquéllos sobre Base de Datos, mientras uno de Ingeniería Civil, no corresponde a ninguna de las clases consideradas en el ejemplo.

4.3.1.5. Relaciones n-arias

Asociaciones generales pueden ser modeladas, mediante **relaciones**. Estas pueden ser de aridad múltiple. Estas asociaciones comprenden el concepto de interacción entre objetos con semántica particular, para cada sistema considerado.

Estas relaciones pueden también poseer características inherentes a ellas (cardinalidad, atributos), producto de la asociación entre los objetos.

> **Ejemplo 4.3.1.5:** En un ambiente de procesamiento de datos, puede interesar conocer que usuarios accesa alguna aplicación, residente en algún equipo, a través de algún terminal. Esto se puede representar como una relación cuaternaria entre terminales, aplicaciones, usuarios y máquinas.

3.3.2. Definición de tipos por Intensión.

La mayoría de los conceptos que maneja el ser humano se definen por intensión, más que por extensión. Es decir, se tiene una descripción genérica de las condiciones que debe satisfacer un objeto para ser representante de un concepto. En modelos de datos tradicionales, se ha dado cierta preferencia a la definición de conceptos por extensión; esto posiblemente como producto de la influencia del contexto dentro del cual surgieron tales modelos, en el cual la orientación era estandarizar la representación y manipulación de datos corporativos.

Un caso diferente es el de los modelos semánticos, a cuyo surgimiento está asociado el deseo de representar de la manera más fiel y natural posible el conocimiento humano; este entorno resulta claramente propicio para preferir la definición de conceptos por intensión. El modelo S.O.O.M. soporta y favorece definiciones por intensión, mediante aserciones sobre los objetos que sobre él se definen.

4. EL MODELO S.O.O.M.

El producto de la selección de las características que debería tener un Modelo de Datos que se hizo en el capítulo anterior es el Modelo S.O.O.M., que se presenta a continuación, describiendo, ante todo, los conceptos y abstracciones que soporta, para luego definirlo más formalmente y mostrar la representación gráfica que se propone para su utilización en la modelación de casos reales.

4.1. Conceptos y Abstracciones Soportadas por el Modelo

El modelo soporta tres conceptos fundamentales: **clase**, **relación** y **jerarquía de clases**.

Clase: Un conjunto de entes con propiedades comunes, constituye un tipo o **clase**. Los entes que conforman una clase son llamados **objetos**. Una clase puede ser: un **tipo concreto**, esto es, predefinido por el ambiente (p.ej., enteros), o un **tipo abstracto**, constituido como abstracción de algún conjunto, compuesto o nominal de tipos. A cada clase se asocia un conjunto de **restricciones**, las cuales son explícitamente definidas a momento de modelación.

Relación: Una relación es, en términos matemáticos, un subconjunto del producto cartesiano de los dominios. Diremos que una relación es una **asociación** entre un conjunto de diversas clases (los dominios).

Una relación puede estar **parametrizada**, esto es, tiene atributos que no son propios de los objetos que asocia, sino de la asociación misma. Para cada dominio de una relación, se define su grado de participación en ella: **restricciones de cardinalidad**, en la cual se denotan las cotas inferior y superior para el número de objetos de esa clase (dominio) que pueden estar relacionados en una tupla de objetos del resto de los dominios. Otras restricciones: **aserciones** sobre los objetos participantes en la relación, pueden también asociarse a la relación, para denotar su **semántica**.

Jerarquía de Clases: Se puede establecer **jerarquías** entre clases, esto es, una clase puede tener propiedades (atributos o relaciones) que la identifican con otras clases. Esto se abstrae con los conceptos de **subclases** y **superclases**: diremos que una clase es una subclase de otra, la cual llamaremos superclase, si hay una jerarquía de la segunda a la primera.

Sobre la jerarquía de clases de establece un mecanismo que asocia a los objetos de la subclase, propiedades de la superclase, o viceversa: **herencia**. Se establecen tipos de herencia, según tipos de jerarquía. Cada tipo de jerarquía corresponde a una de las abstracciones presentadas en el Capítulo 3, y son:

Generalización: un conjunto de clases con características comunes (semánticas) puede abstraerse, mediante generalización, en una clase genérica, que puede ser atribuida por tales características. Esta abstracción establece un esquema de **herencia hacia arriba**, es decir, la clase resultante de la abstracción, hereda aquellas propiedades comunes que permiten la generalización.

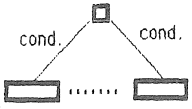
Clasificación: Los objetos de una clase pueden ser agrupados en subclases **disjuntas**, mediante un conjunto de atributos que los discriminan. En una clasificación los objetos de la subclase heredan los propiedades de la superclase, esto es, **herencia hacia abajo**.

Tipificación. Dentro de una clase podemos observar objetos con características comunes que los agrupan, no necesariamente en forma disjunta. La abstracción que soporta tales agrupaciones es la **tipificación**. La tipificación induce un esquema de herencia hacia abajo.

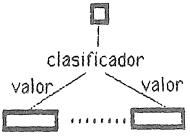
2.4. Representación Gráfica de las Abstracciones y Conceptos soportados por el Modelo S.O.O.M..

Conceptos:

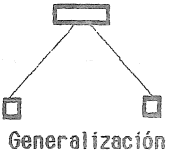
Abstracciones:



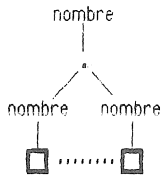
Tipificación



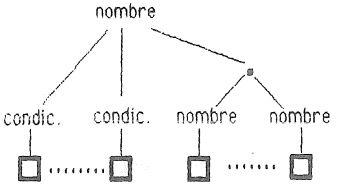
Clasificación



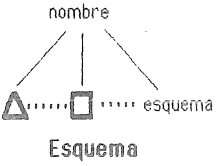
Generalización



Tipo Abstracto



Relación



Esquema

3.3. Ejemplo de modelación en S.O.O.M.

Para ejemplificar el uso de S.O.O.M. en la modelación de una Base de Datos, se presenta la Definición formal de un Diccionario de Datos para un Sistema de Bases de Datos que soporte el modelo. Esta definición forma parte de la implementación de un lenguaje de definición de datos para el modelo S.O.O.M. realizada en el proyecto de grado en el que fue definido del modelo que aquí se presenta.

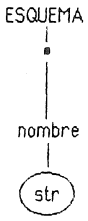
3.3.1 Esquema en S.O.O.M. de un Diccionario de Datos.

Un diccionario de datos puede ser visto como una Base de Datos, que contiene datos sobre otras Bases de Datos, y su definición puede ser vista como la de un esquema.

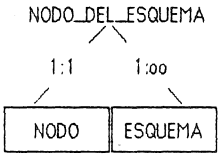
Atendiendo a esta consideración, la definición del Diccionario de Datos, corresponde a la de un esquema en el modelo S.O.O.M. La definición se presenta utilizando la representación gráfica del modelo presentada en la sección anterior.

Esquema de la Visión del DDL del DD

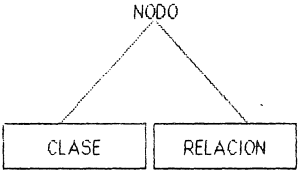
Esquemas definidos como:



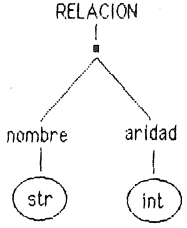
cuyos nodos son:

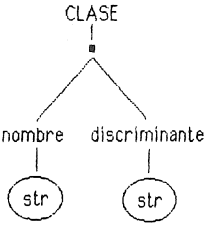


siendo **nodo** una generalización de clases y relaciones:

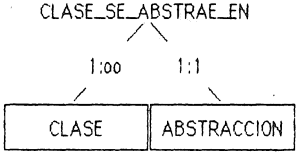


con:

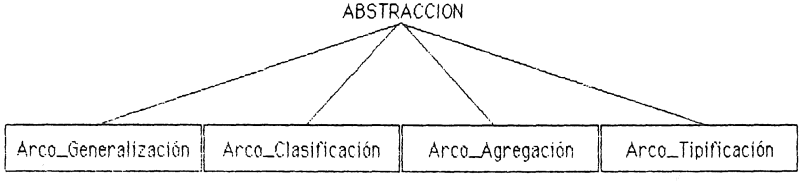




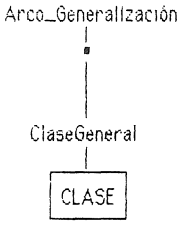
a cada **clase** se asocian:

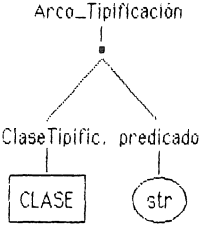


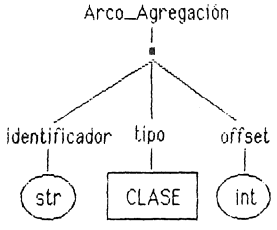
donde:

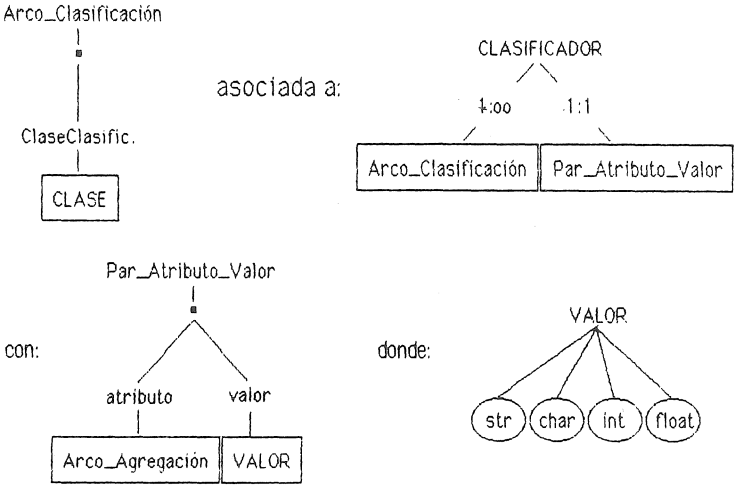


con:

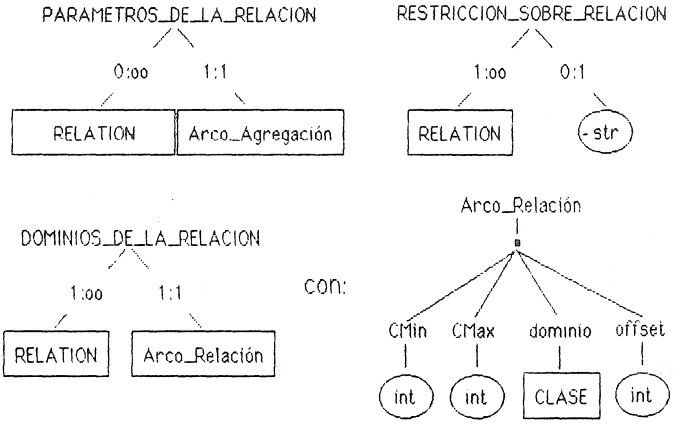








A cada relación se asocian:



CONCLUSIONES

Realizado el análisis de los principios básicos que debe cumplir un modelo (expresividad y economía conceptual), y una revisión de los paradigmas de modelación existentes en bases de datos, se definió un modelo de datos enmarcado dentro del paradigma semántico de modelación, el modelo SOOM.

El modelo S.O.O.M., aquí propuesto, meneja los conceptos de: clase (conjunto de objetos con características comunes) y relación (asociación entre clases); soporta las abstracciones: agregación (construcción de clases de objetos compuestos), generalización (clase que agrupa varias clases con semántica

común, clasificación (clases resultantes de la discriminación de objetos de una clase) y tipificación (clases resultantes de agrupar objetos pertenecientes a una clase con características particulares no excluyentes).

Presentamos el modelo S.O.O.M. como un modelo semántico porque: 1) distingue clases de relaciones; 2) provee abstracciones necesarias para la creación de nuevos conceptos a partir de los ya existentes; 3) favorece la definición de conceptos por intensión, mediante el uso de restricciones sobre clases y relaciones que denotan la semántica de éstas.

Además, el modelo S.O.O.M., ha sido enriquecido con la introducción de los conceptos de jerarquía y herencia entre clases (programación orientada por objetos), según las abstracciones de generaclización, clasificación y tipificación.

REFERENCIAS

Ballou, N, Chou, H-T., Garza, J.F., Woelk, D. "Features of the Orion Object Oriented Database System" en "Object Oriented Concepts, Databses and Applications", Part 3, Cap. 12, pp. 251-282. ACM Press, New York, 1.989.

Beech, David. "Intensional Concepts in an Object Database Model". ACM OOPSLA Proceedings, Sept. 88.

Bloom, T. y Zdonik, S. "Issues in the Design of Object-Oriented Database Programming Languages". ACM OOPSLA Proceedings, Oct. 87.

Bretl, R, Maier, D., Otis, A., Penney, J., Schuchardt, B, Stein, J, Williams, E.H., Williams, M. en "Object Oriented Concepts, Databses and Applications", Part 3, Cap. 12, pp. 283-308. ACM Press, New York, 1.989.

Brodie, M.L. "On the development of data models", en "On Conceptual Modelling Perspectives from Artificial Intelligence, Databases and Programming Languajes, Brodie, Mylopoulos, Schmidt, Ed. Springer-Verlag, Nueva York, 1.984. pp. 19-48

Caruso, M. y Zdonik, S. Report on "Object-Oriented Database Workshop: Semantic Aspects" ACM OOPSLA Addendum to the Proceedings, Oct. 1987.

Chen, P. "Then entity-relationship model: Toward a unified view of data", ACM Trans. Database Syst. 1. pp. 9,36

Codd, E.F.,. Extending the Database relational model to capture more meaning". ACM Trans. Database Syst., Vol. 4, Nº 4, Dic. 1.979, pp. 397-434.

Date, C.J. An Introduction to Database Systems, Vol. II. Addison-Wesley Publishing Company, 1.984.

Hammer, M, y McLeod, D. "Database Description with SDM: A semantic Database Model, ACM. Trans. Database Syst., Vol 6, Nº 3, Sept. 1981, pp. 351-386.

Kim, W y Lochovsky, F. Edit. "Object Oriented Concepts, Databses and Applications" ACM Press, New York, 1.989

Kim, W., Ballou, N., J., Chou, H., Garza, J. y Woelk, D. "Integrating an Object-Oriented Database Programming System with a Database System". ACM OOPSLA Proceedings, Sep. 88.

Kim, W., Banerjee, J., Chou, H., Garza, J. y Woelk, D. "Composite Object Support in an Object-Oriented Database System". ACM OOPSLA Proceedings, Oct. 87.

King, R. y MacLeod, D. "The Event database specification model". Proceedings of 2nd. International Conference on Database: Improving Usability and Responsiveness. (Jerusalem, Israel), 1.986, pp. 229-322.

Maier, D., Stein, J., Otis, A., y Pardy, A.. "Development of an Object-Oriented DBMS", ACM OOPSLA Proceedings, Sept. 86.

Nierstrasz, O. "A survey on Object Oriented Concepts", en "Object Oriented Concepts, Databses and Applications", Part I, Cap. 1, pp. 3-32. ACM Press, New York, 1.989.

Nixon, B, Chung L., Lauzen, I., Borgida A., Mylopoulos, J y Stanley, M. "Implementation of a compiler for a semantic data model: Experience with Taxis". Proceeding of the ACM SIGMOD Conference (San Francisco), ACM, 1.987.

Peckham J. y Maryansky, F. "Semantic Data Models". ACM Computing Surveys, V. 20, #3, Sept. 88.

Penney, D.J. y Stein, J.. "Class Modification in the GemStone Object-Oriented DBMS". ACM OOPSLA Proceedings, Oct. 87.

Shipman, D.W., "The Functional data model and the data languaje DAPLEX", ACM Trans. Database Syst., Vol. 6, Nº 1, Marzxo 1.981, pp. 140-173.

Smith, J.M. y Smith, D.C.P., "Database Abstractions: Aggregation and generalization", ACM Trans. Database Syst. 2, 1.977 (b) pp. 105-133.

Smith, J.M. y Smith, D.C.P., "Databse Abstractions: Aggregation", Commun. ACM. Nº 20, 1.977 (a), pp. 405-413.

Sundgren, B. "Conceptual Foundation of the infologic approach to data bases", Data Base Management, North Holland, Amsterdam, pp. 61-96, 1.974

Teorey, T., Yang, D. y Fry, J.P.. "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model". ACM Computing Surveys, Vol. 18, No. 2, Jun. 1.986.

Thatte, S. Report on "Object-Oriented Database Workshop: Implementation Aspects". ACM OOPSLA Addendum to the Proceedings, Oct. 1987.

Tsichritzis, D. y Lochovsky, F. Data Models, Prentice-Hall, 1.982.

Search in Horn's knowledge bases and the Simplex algorithm

by

Julián Aráoz

Departamento de Procesos y Sistemas
Universidad Simón Bolívar

ABSTRACT

It has been proved that deduction in propositional Horn's clauses systems is equivalent to feasibility of a linear program associated to a directed hyper graph flow model which correspond to the Horn's clauses systems. Here, we show that bottom up search is equivalent to solve the linear program using the Simplex algorithm and top down search correspond to the application of the dual Simplex algorithm.

1. Introduction

In the last few years, expert systems [7,9,17] have become one of the most important areas in artificial intelligence. This technology has been developed for a wide variety of practical problems resolution, and has been established as a new programming methodology paradigm. In this way, the knowledge used building a system transcends program limits and become part of the data. That is why expert systems are also called knowledge-based systems.

To represent a given knowledge base various schemes such as associative networks [11], frames [14], rule based models [7,16], and many others, are used. Such a knowledge base is usually modifiable by user interaction and is handled by the system in order to produce new knowledge or to check if a given fact is

a consequence of the represented knowledge. Generally, these operations have been seen as a particular case of logical deduction and have been treated with logical methods.

From other point of view, any knowledge representation model is essentially a combinatorial data structure, since it can be viewed as a discrete set with some defined combining rules. Now then, a deduction is also a search in such structure. This leads to the formulation of models for deduction in knowledge-based systems taken from the combinatorial optimization area. Even so, there are no publications about this use of combinatorial models, but the set-covering model used for abductive systems by Reggia, Nau, Wang and Peng [21,24,25,26]. However, since this area has grown very fast in last twenty years [19,23], we believe that the systematic use of this approach will be useful for improving the efficiency of inference engines, and will establish a positive exchange between the two disciplines.

In this paper, we deal with propositional Horn clauses. This is not the most general knowledge representation scheme, but an expressive and widely used one. A directed hypergraph based model for Horn clauses systems, which leads to the equivalence between logical deduction in such systems and a linear programming problem corresponding to a generalization of flow to directed hypergraphs has been proposed [29]. In this paper we show that bottom up search is equivalent to solve linear program using the Simplex algorithm with artificial variables and top down search correspond to the application of the dual Simplex algorithm.

2. Paths and flows in directed hypergraphs

In order to work with the dual of linear problems formulated on the incidence matrix of a graph, Berge [5] introduced the definition of hypergraph, a generalization of the concept of graph, which consist of a set of vertices and a family of hyperedges, each one a subset of vertices with cardinality not restricted to be equal to two (as it is on graphs). In this way any matrix with only zero or one valued entries is the incidence matrix of a hypergraph and so it is its transpose. Usually empty hyperedges are undesired, and isolated vertices

are also forbidden, in order to assure that the dual holds the same property.

The same idea may be trivially extended to directed graphs, leading to the concept of directed hypergraphs. This combinatorial model may be useful for modelling in a series of applications where a generalization of directed graphs is needed. For example, directed hypergraphs have been used as an expressive representation for functional dependencies in databases [2,28], Petri nets [22], and associative networks [6]. On the other hand, most of combinatorial problems and some algorithms on directed graphs can also be extended to directed hypergraphs.

Definition 2.1: Let V be a finite nonempty set, and let $E \subseteq \mathcal{P}(V)^2$. Then the pair $H = (V, E)$ is a *directed hypergraph*, iff $\forall\, e \in E$ such that $e = (t(e), h(e))$:

i) $t(e) \cap h(e) = \emptyset$,

ii) $t(e) \cup h(e) \neq \emptyset$.

An element of V will be called a *vertex* and an element of E will be called a *hyperedge* or simply an *edge* when not leading to confusion. Henceforth, the set of all vertices of a directed hypergraph H will be denoted as V_H and the set of all its edges as E_H, when not leading to confusion the subindex H will be omitted. If $e = (V', V'')$ is an edge of a directed hypergraph, V' will be called the *tail* of e, and V'' the *head* of e. The tail of an edge e will also be denoted as $t(e)$ and its head as $h(e)$.

Definition 2.2: Let H be a directed hypergraph. Then the *incidence matrix* of H will be $\overline{H} = (h_{v,e} : v \in V; e \in E)$, where:

$$h_{v,e} = \begin{cases} -1 & \text{if } (v \in t(e)) \\ 1 & \text{if } (v \in h(e)) \\ 0 & \text{if } (v \notin t(e) \cup h(e)). \end{cases}$$

Definition 2.3: An *increasing path* from s_0 to s_k $(k \geq 0)$ in a directed hypergraph H is a sequence of form:

$$s_0 e_1 s_1 e_2 s_2 \ldots s_{k-1} e_k s_k$$

where

i) $\forall\, i \in [0 \ldots k]\ (s_i \in \mathcal{P}(V))$,

ii) $\forall\, i \in [1 \ldots k]\ (e_i \in E)$,

iii) $\forall\, i \in [1 \ldots k]\ (t(e_i) \subseteq s_{i-1})$,

iv) $\forall\, i \in [1 \ldots k]\ (s_i = s_{i-1} \cup h(e_i))$.

Now we present a generalization of the problem of flow in networks [15] to directed hypergraphs. Most of notation and definitions are extended from Rockafellar [27].

Definition 2.4: Let H be a directed hypergraph. Then a *flow* F in H is just a function of the edges of H into the real numbers, i.e. $F : E \to \mathbb{R}$. The value $F(e)$ will be called the *flux* of the edge e. A flow F is an *integral flow* iff for all e in E, the flux of e is integer, that is $\text{Range}(F) \subset \mathbb{N}$. For a flow F, the set of edges with positive flux will be denoted as $P(F)$, i.e. $P(F) = \{e \in E : F(e) > 0\}$. A flow F will be called a *nonnegative flow* iff it has nonnegative flux for all edges.

Definition 2.5: Let H be a directed hypergraph and let F be a flow in H. The *divergence function* of F is $\text{Div}_F : V \to \mathbb{R}$, where:

$$\text{Div}_F(v) = \sum_{\{e \in E : v \in h(e)\}} F(e) - \sum_{\{e \in E : v \in t(e)\}} F(e).$$

Note that divergence function for v_i can be calculated by multiplying ith row of the incidence matrix of H by the vector F. Therefore:

Lemma 2.6: *Let H be a directed hypergraph and F a flow in H, then:*

$$\overline{H} * \overline{F} = \overline{\text{Div}_F}. \qquad \square$$

Since for the scope of this paper only nonnegative flows are relevant, thereafter whenever a flow appears it will be assumed to be a nonnegative flow.

Definition 2.7: Let H be a directed hypergraph, V' and V'' disjoint subsets of V, and let F be a flow in H. Then F is an *unit flow* from V' into V'' iff:

i) $\forall\, v \in V''\ (\text{Div}_F(v) = 1)$,

ii) $\forall\, v \in V - (V' \cup V'')\ (\mathrm{Div}_F(v) = 0)$,

iii) $\forall\, e \in E\ (F(e) \geq 0)$.

Therefore F is an unit flow if and only if it is a feasible solution to the linear program defined in 2.7.

3. Horn systems and rule hypergraphs

Even when a variety of knowledge representation schemes exist [7,11,14,16], perhaps the most precise and general model is given by symbolic logic. In this way a knowledge base is simply a well formed formula in a formal logic system. It is known that any logical formula can be translated to an equivalent one in clausular form [20]. In a propositional system a clause is nothing more than a statement of the form: $\bigwedge_{\{p \in P\}} p \rightarrow \bigvee_{\{p' \in P'\}} p'$, where P and P' are both sets of atomic propositions, i.e. instances of propositional variables. A formula is in clausular form if it is a finite conjunction of clauses.

A particular class of clauses, very important for the simplicity of the derivations with them and their suitableness for expressing most of knowledge, are Horn clauses [18] in which there is only one atomic proposition at the right side, i.e. clauses of the form: $\bigwedge_{\{p \in P\}} p \rightarrow p'$. If a formula is a finite conjunction of Horn clauses it's said that it is in Horn clausular form. Note that a clause of the form $\bigwedge_{\{p \in P\}} p \rightarrow p'$, where $p' \in P$ is a tautology, so any formula in Horn clausular form can be assumed to have not such clauses.

Definition 3.1: Let P be a set of atomic propositions. Then a *Horn system* S is a pair (P, C), where C is a set of nontautological Horn clauses referencing only the atomic propositions in P.

A Horn system is then naturally modelled by a directed hypergraph with all edges having a unitary head. In this model vertices represent the atomic propositions of the system and edges represent the clauses.

Definition 3.2: A directed Hypergraph H is a *rule hypergraph* iff:

$$\forall\, e \in E(|h(e)| = 1).$$

Any edge of H will be called a *rule* and any vertex an *atom* of H. A rule of the form $(T, \{v_h\})$ will be denoted as $T \rightarrow v_h$. Any subset of V will be an *information state* of H.

Then every atomic proposition p of a logical system will be represented by an atom v_p and every clause of the form: $\bigwedge_{\{p \in P\}} p \rightarrow p'$ where $(p' \notin P)$ by a rule $e = V_p \rightarrow v_{p'}$, where $V_p = \{v_p : p \in P\}$. In this way any Horn system S can be modelled by a rule hypergraph, that will be denoted $H(S)$.

Fagin [12,13] have established that derivation in Horn systems is equivalent to functional dependencies [1] derivation in a relational database model [8], that is, Fagin showed that a Horn clause is logically implied by a given set of Horn clauses iff the same holds interpreting each clause as a functional dependency. In other works [2,3,4,28], it's been established that derivation of functional dependencies is also equivalent to the determination of membership of a clause to the closure of the corresponding directed hypergraph.

Now, we introduce a path-based concept of derivation which is equivalent to deduction in Horn's systems [29].

Definition 3.3: Let H be a rule hypergraph. A *derivation* of d from O in H, where d is a vertex and O an information state, is an increasing path, of the form:

$$s_0 e_1 s_1 e_2 s_2 \ldots s_{k-1} e_k s_k$$

where:

i) $s_0 = O$,

ii) $d \in s_k - O$.

The edge sequence of a derivation will be called *rule sequence*. A rule contained in the rule sequence of a derivation is said to be *applied* in it. A rule e is said to be *applicable* given an information state O iff $t(e) \subseteq O$.

Note that the concept of derivation in rule hypergraphs corresponds to that of logical derivation with Horn clauses by *modus ponens*. If information states are interpreted as the conjunction of the propositions represented by its atoms, which are assumed to be true, then a rule is applicable if its preconditions are true, and the set of true propositions after the rule is applied is augmented by its consequent.

Note also that the increasing path correspond to a bottom up search in the Horn's system.

Theorem 3.4 [29]: *Let $S = (P, C)$ be a Horn's system with an information state O and an atom d. The vertices of $H(S)$ are P and the edges of $H(S)$ are C. Let Q be the set of solutions F to the linear program*

$$\begin{aligned} &\mathrm{Div}_F(d) = 1. \\ &\forall\ v \in P - (O \cup \{d\})\ (\mathrm{Div}_F(v) = 0), \text{ and} \\ &\forall\ e \in C\ (F(e) \geq 0). \end{aligned}$$

Then, the following statements are equivalent:

i) d is deductible from O in S;

ii) There is a derivation of d from O in $H(S)$;

iii) There exists an integral F in Q;

iv) Q is not empty. □

4. Bottom up search and the Simplex algorithm

In this section we will show that the Simplex algorithm with artificial variables applied to the system Q in Theorem 3.4 behaves exactly as a bottom up search. Hence bottom up search could be considered a simplex algorithm which take advantage of the structure of the problem in the same way that the transportation algorithm does.

Theorem 4.1: *Bottom up search is a Simplex algorithm.*

Proof. We prove the theorem by showing, using induction, that the derivation of d from O,

$$s_0 e_1 s_1 e_2 s_2 \ldots s_{k-1} e_k s_k$$

is obtained with $s_0 = O$ and at any pivoting step t, e_t is the edge corresponding to the pivot column. This is equivalent to prove that at step t we can pivot in a column only if all the preconditions of this column are in s_{t-1}.

Note that after adding the artificial variables, the problem is to minimize the summation of the artificial variables. At any pivoting step, a variable j could enter the basis only if its transformed cost $z_j - c_j > 0$, i.e. $z_j > 0$ since $c_j = 0$ for the real variables.

Let $z_j = \sum_{i \in I} h_{ij}$ with $I \subseteq P$. Then $z_j > 0$ is equivalent to $z_j = 1$ and $h_{ij} = 0$ for all $i \in I$ but one i' with $h_{i'j} = 1$, because h_{ij} is equal to 0, 1 or -1 with exactly one 1. That is $z_j > 0$ iff all preconditions of j are in $P \setminus I$.

(i) For $t = 1$, the basis is the identity matrix, hence $z_j = \sum_{i \in P-O} h_{ij} > 0$ iff all preconditions of j are in $O = s_0$.
Without loss of generality assume that $z_e = 1 = h_{e1}$. Hence, after pivoting, we have a new basis $\begin{vmatrix} 1 & O \\ O & I \end{vmatrix}$ where $\begin{vmatrix} 1 \\ O \end{vmatrix}$ is the column of edge e and $\begin{vmatrix} O \\ I \end{vmatrix}$ are the columns of the artificial variables. The path is now $Oe(O \cup \{h(e)\})$.

(ii) Assume that at pivoting step t we have built the path $s_0 e_1 s_1 \ldots e_{t-1} s_{t-1}$ and we have rearranged the basis as $\begin{vmatrix} B & O \\ O & I \end{vmatrix}$ where $\begin{vmatrix} B \\ O \end{vmatrix}$ are the columns of $e_1, \ldots, e_{t-1}$ and $\begin{vmatrix} O \\ I \end{vmatrix}$ are the columns of the artificial variables. Hence the inverse of the basis is $\begin{vmatrix} B^{-1} & O \\ O & I \end{vmatrix}$ and $z_j = \sum_{i \in P - s_{t-1}} h_{ij}$.
We have $z_j > 0$ iff all the preconditions of j are in s_{t-1}. Moreover, if we pivot in $z_j > 0$, since $h_{ij} = 0$ for the remaining artificial variables we keep the same structure for step $t + 1$. □

It is not difficult to show that in Theorem 3.4 we can change the ='s by ≥'s and the Theorem is still valid. If we add the surplus variables we have an initial basis with the surplus variable corresponding to d equals to -1. If we apply the dual Simplex method, it will behave like a top down search.

References

[1] Armstrong, W.W.: Dependency structures of data base relationships. Information Processing Letters, Vol. 74, 1974, pp. 580–583.

[2] Ausiello, G., D'Atri, A., Saccà, D.: Graph algorithms for functional dependency manipulation. Journal of the ACM, Vol. 30, No. 4, October 1983, pp 752–766.

[3] Ausiello, G., D'Atri, A., Saccà, D.: Minimal representation of directed hyper-

graphs. SIAM Journal of Computing, Vol. 15, No. 2, May 1986, pp. 418–431.
[4] Beeri, C., Bernstein, P.A.: Computational problems related to the design of normal form relational schemes. ACM Transactions on Database Systems, Vol. 4, 1979, pp. 30–59.
[5] Berge, C.: Graphes et hypergraphes. Second edition, Dunod, Paris, 1976.
[6] Boley, H.: Directed recursive labelnode hypergraphs: A new representation language. Artificial Intelligence, Vol. 9, 1977, pp. 49–85.
[7] Buchanan, B.G., Shortliffe, E.H. (Editors): Rule-based expert systems. Addison-Wesley, Massachusetts, 1984.
[8] Codd, E.F.: A relational model of data for large shared data banks. Communications of the ACM, Vol. 13, No. 6, 1970, pp. 377–387.
[9] Charniak, E., McDermott, D.: Introduction to artificial intelligence. Addison-Wesley, Massachusetts, 1986.
[10] Dantzig, G.B.: Linear programming and extensions. Princeton University Press, New Jersey, 1963.
[11] Deliyanni, A., Kowalski, R.A.: Logic and semantic networks. Communications of the ACM, Vol. 22, No. 3, 1979, pp. 184–192.
[12] Fagin, R.: Functional dependencies in a relational database and propositional logic. IBM Journal of Research and Development, Vol. 21, No. 6, November 1977, pp. 534–544.
[13] Fagin, R.: Horn clauses and database dependencies. Proceedings of the 12th Annual ACM Symposium on the Theory of Computing, California, 1980, pp. 33–64.
[14] Filkes, R., Kehler, T.: The role of frame-based representation in reasoning. Communications of the ACM, Vol. 28, No. 9, 1985, pp. 904–920.
[15] Ford, L.R., Fulkerson, D.R.: Flows in networks. Princeton University Press, New Jersey, 1962.
[16] Hayes-Roth, F.: Rule-based systems. Communications of the ACM, Vol. 28, No. 9, 1985, pp. 921–932.
[17] Hayes-Roth, F., Waterman, D.A., Lenat, D. (Editors): Building expert systems. Addison-Wesley, Massachusetts, 1983.
[18] Horn, A.: On sentences which are true of direct unions of algebras. Journal of Symbolic Logic, Vol. 16, No. 14, 1951.
[19] Lawler, E.L.: Combinatorial optimization: Networks and matroids. Holt, Rinehart and Winston, 1984.
[20] Manna, Z.: Mathematical theory of computation. McGraw-Hill Inc., 1974.
[21] Nau, D.S., Reggia, J.A.: Relationships between deductive and abductive inference in knowledge-based diagnostic problem-solving. Expert Database systems, The Benjamin/Cummings Publishing Company, 1986, pp. 549–558.
[22] Petri, C.A.: Communication with automata. Tech. Rep. RADC-TR-65377, Vol. 1, Griffith Air Force Base, New York, 1966.
[23] Papadimitriou, C.H., Steiglitz, K.: Combinatorial optimization. Algorithms and complexity. Prentice-Hall Inc., New Jersey, 1982.
[24] Reggia, J.A., Nau, D.S., Wang, P.Y.: Diagnostic expert systems based on a set covering model. International Journal of Man-Machine Studies, Academic Press Inc., London, 1983, pp. 437–460.
[25] Reggia, J.A., Nau, D.S., Wang, P.Y.: A formal model of diagnostic inference. I. Problem formulation and decomposition. Information sciences, Vol. 37, 1985, pp. 227–256.
[26] Reggia, J.A., Nau, D.S., Wang, P.Y., Peng, Y.: A formal model of diagnostic in-

ference. II. Algorithmic solution and application. Information Sciences, Vol. 37, 1985, pp. 227–256.

[27] Rockafellar, R.T.: Network flows and monotropic optimization. John Wiley and Sons, 1984.

[28] Saccà, D.: Closures af database hypergraphs. Journal of the ACM, Vol. 32, No. 4, October 1985, pp. 774–803.

[29] Torres, A., Aráoz, J.: Combinatorial models for searching in knowledge bases. Acta Científica Venezolana, Vol. 39, 1988, pp. 387–394.

FORMAS NORMALES MODELO ENTIDAD RELACION.

Alonso Márquez y Maria Esther Vidal.

Universidad Simón Bolívar.

Caracas-Venezuela.

RESUMEN

Uno de los Modelos de Datos más ampliamente usado en la etapa de diseño conceptual debido a su legibilidad, expresividad y economía conceptual, es el Modelo Entidad Relación.

Pese a esto, no existe una definición formal satisfactoria que permita interpretar de una manera no ambigua un esquema E-R. Tampoco existe una Teoría de Normalización que permita decidir si un esquema E-R es bueno, es decir, está libre de las anomalías de inserción, modificación y eliminación.

En este articulo, se presenta una especificación formal del Modelo E-R a partir de la cual se definen formas normales para esquemas E-R. Los esquemas E-R que cumplen estas formas normales tienen la propiedad que el esquema equivalente en Relacional, cumple como mínimo 3FN.

Además, se presentan técnicas de traducción de esquemas E-R a esquemas E-R normalizados y técnicas de traducción de esquemas E-R a esquemas Relacionales + Restricciones de Integridad que representan toda la semántica del esquema E-R.

1. INTRODUCCION.

Una base de datos es una representación de los datos operativos de una organización[8].

Tradicionalmente en el área de Base de Datos se denomina Modelo de Datos a la herramienta de modelaje de datos o sistema formal de especificación que permite representar las propiedades y características de los datos operativos de una organización, es decir, es la herramienta utilizada para definir la base de datos[19].

El resultado de la utilización de un modelo de datos para especificar una situación en particular se denomina esquema[19].

Dado un modelo de datos y una situación particular, existen muchos esquemas que representan dicha situación. Dependiendo de que tanto se acerque a la realidad, se puede clasificar el esquema en bueno o malo.

Luego sería de gran interés, conocer las características de los buenos esquemas de un modelo de datos particular y técnicas para transformar esquemas malos en buenos.

La teoría que engloba todas estas técnicas se denomina Teoría de Normalización, donde el nombre viene de suponer que un esquema se considera "bueno" si cumple una cierta forma normal.

Para el único modelo de datos que ha sido definida una Teoría de Normalización es para el modelo Relacional[5,6,20]. Este modelo debido a las abstracciones que ofrece, no es utilizado como herramienta de especificación conceptual. Por el contrario, el modelo de datos más ampliamente usado es el modelo Entidad Relación[7,12,16]. A pesar de esto, no existe definida una teoría de normalización para el mismo.

Por esta razón, se está realizando un trabajo que tiene como objetivos fundamentales dar una definición formal del modelo E-R

y sobre ésta, definir uná Teoría de Normalización.

En este artículo serán presentados algunos de los logros alcanzados en este trabajo: elementos de la especificación formal del modelo, formas normales de esquemas E-R, técnicas de normalización y técnicas de traducción a esquemas relacionales.

Para dar la especificación del modelo Entidad Relación se utilizará el Lenguaje de la Lógica de Predicados y conceptos de la Teoría de Conjuntos. Las formas normales definidas poseen la propiedad que el esquema equivalente en relacional cumple como mínimo la 3FN.

Las técnicas de normalización de esquemas E-R presentadas cumplen la propiedad que en el esquema resultante se mantiene toda la información.

Por otro lado, las técnicas de traducción cumplen la propiedad que toda la información representada en el esquema E-R también se representa en el Relacional, ya que aquellas abstracciones que no tienen abstracción equivalente son representadas a través de restricciones de integridad.

Para presentar estos logros el resto de este artículo se estructura de la siguiente manera: en la sección II se muestran los antecedentes del problema presentado. En la sección III se da una especificación del Modelo E-R. En la sección IV se definen las Formas Normales de un esquema E-R. En la sección V se presentan técnicas de normalización. En la sección VI se presentan técnicas de traducción a Relacional. Finalmente, en la sección VII se presentan las conclusiones y posibles extensiones.

2. ANTECEDENTES.

Son bien conocidas las anomalías de inserción, modificación y eliminación que se pueden presentar en una Base de Datos como consecuencia de no realizar un buen diseño de la misma[6,20]. Para evitar estos problemas se pueden adoptar alguna de las siguientes soluciones:

a) Utilizar un modelo de datos que permita representar restricciones de integridad, es decir, representar cuales son los estados de la base de datos que no son permitidos.

b) Diseñar esquemas que cumplan ciertas características, es decir, que satisfagan una cierta forma normal.

La solución más ampliamente usada y estudiada es la de llevar el esquema hasta una cierta forma normal.

En esta dirección, se ha definido una Teoría de Normalización para el modelo de datos Relacional.

El modelo de datos Relacional[5,6,8,20], es un modelo que ofrece la abstracción de relación, siendo una relación un subconjunto del producto cartesiano de los dominios sobre los cuales la misma está definida. Una relación puede ser vista como una tabla donde las columnas representan los atributos de la relación y, las filas o tuplas, las instancias u ocurrencias de ésta.

Las restricciones que se permiten representar a través del modelo son de dependencia funcional y multivaluada entre atributos y las inherentes a éste, identidad e integridad referencial.

Dado que ofrece una única abstracción, el modelo presenta las siguientes características: economía conceptual, sobre carga

de abstracciones y poca expresividad. Un ejemplo donde se muestran las limitaciones expresivas del mismo es el siguiente:

Representar a través de un esquema relacional las materias que ha inscrito en un trimestre un estudiante cumpliendo la restricción que como mínimo puede inscribir 2 materias y como máximo 5.

Las relaciones EST y MAT mantienen la información de los estudiantes y de las materias respectivamente.

EST(CARNET,NOMBRE) CLAVE:CARNET MAT(COD-MAT,N-MAT) CLAVE:COD-MAT

Por otro lado, la relación INSCRIBE mantiene la información de cada estudiante con las materias que inscribe.

INSCRIBE(COD-MAT,CARNET) CLAVE: COD-MAT,CARNET.

CLAVES FORANEAS: COD-MAT NOT NULL; CARNET NOT NULL.

Pero, la restricción de que dado un estudiante puede tener asociado como mínimo dos materias y como máximo cinco, no se puede representar a través de las estructuras del modelo.

Por causa de su poca expresividad, el modelo Relacional no se utiliza como herramienta de especificación en la etapa de diseño conceptual. Por el contrario, para esta etapa se utilizan modelos de datos que permiten representar de una manera más fiel la situación a modelar. Por esta razón, se utilizan los modelos de datos semánticos, los cuales son herramientas de especificación que permiten representar gran parte de las abstracciones que existen en el mundo real. Entre los modelos semánticos, el más utilizado es el modelo Entidad Relación.

A causa de su economía conceptual, el modelo Relacional es el modelo de datos que más manejadores comerciales ofrecen, es

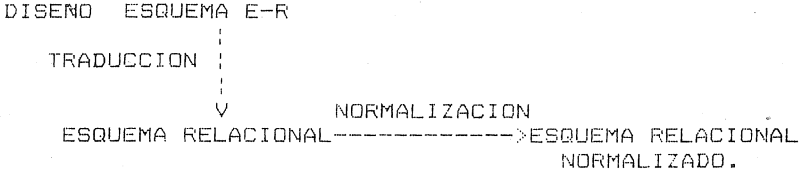
decir, es el modelo que más se utiliza en la etapa de implementación.

Como se dijo anteriormente, existe una Teoría de Normalización para este modelo. En ésta se han definido un grupo de formas normales[5,8,10,20], algoritmos de normalización[0,2,4,9] y criterios de equivalencia de esquemas[20], entre otras cosas.

Las formas normales definidas son seis de las cuales las cuatro primeras imponen restricciones sobre las dependencias funcionales que se pueden cumplir en una relación. De éstas son de interés la 3FN y BCNF, ya que la 2FN es un caso particular de la 3FN y la 1FN impone que las relaciones sean planas.

Considerando 3FN y BCNF lo más razonable de alcanzar es 3FN ya que, existen esquemas relacionales que no pueden alcanzar BCNF sin que exista pérdida de información[8,20]. Además se, ha demostrado que determinar si un esquema relacional tiene un esquema equivalente en BCNF es NP-Completo[3,14].

Dado que solo existe definida una teoría de normalización para el modelo relacional y debido a que éste no se utiliza en la etapa de diseño conceptual , para poder diseñar un esquema normalizado se deben los siguientes pasos:

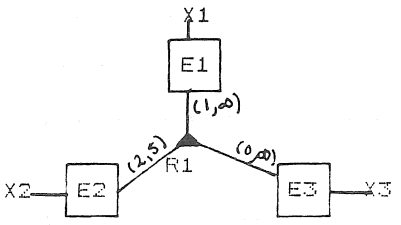


Posponiendo así el proceso de normalización para la etapa de diseño de la implementación, impidiendo de esta manera, en muchos casos, que el diseñador del diseño conceptual pueda verificar si

el esquema normalizado representa lo que realmente se quería representar.

Además de esta limitación, el proceso de normalización a nivel relacional puede presentar problemas. Considérese por ejemplo la siguiente situación:

ESQUEMA E-R ES1



ESQUEMA RELACIONAL ES2

R2(X1,X2,X3)

CLAVE: X1,X2,X3

CLAVES FORANEAS:

X1 NOT NULL, X2 NOT NULL, X3 NULL

DEPENDENCIA FUNCIONAL: X1->X2.

En la traducción del esquema ES1 al esquema ES2 ha habido pérdida de información ya que no se ha podido representar la cardinalidad de E2 en R1.

Siguiendo la definición de 3FN, ES2 no está normalizado. El esquema ES3 es el esquema equivalente normalizado:

ESQUEMA RELACIONAL ES3

R21(X1,X2) CLAVE: X1; R22(X1,X3) CLAVE:X1,X3.

En el proceso de traducción de ES2 a ES3 ha habido pérdida de información ya que, sobre los atributos X1, X2 y X3 de la relación R1 existían restricciones de integridad referencial que en los atributos de R21 y R22 no existen. Esto es debido a que el proceso de normalización se normalizan las estructuras, pero no, las restricciones de integridad. Apartando problemas como éste, ¿qúe ocurriría si en lugar de utilizar el modelo de datos Relacional se utiliza otro modelo de datos para implementar la base de datos?. ¿ Es que las anomalías solo se presentan si no se

hace un buen diseño de esquemas relacionales?.

La respuesta es no. En todo esquema que no esté bien diseñado se presentan estos problemas, luego aunque no se implemente con el modelo relacional, se debería poder garantizar la bondad de la base de datos a implementar.

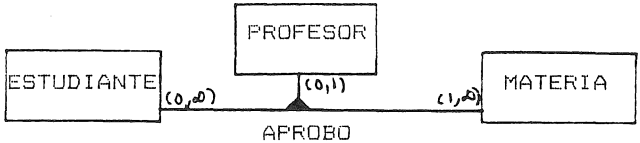
De esta forma, más que tener la teoría de normalización para el modelo Relacional, es necesario tener definida una teoría para un modelo que pueda ser utilizado en la etapa de diseño conceptual, para así garantizar que el esquema conceptual está libre de todos estos problemas.

Como se dijo anteriormente, el modelo E-R es el más ampliamente usado como herramienta de especificación conceptual, luego una posible solución es definir una teoría de normalización para este modelo.

3. MODELO ENTIDAD RELACION.

El modelo Entidad-Relación fue definido de una manera informal en 1976 por Peter Chen[7]. A pesar de su gran aceptabilidad entre los diseñadores de bases de datos y aunque ha sido ampliamente estudiado y analizado en la literatura[15,17,18], no existe una definición formal satisfactoria que permita interpretar un esquema Entidad Relación sin ambiguedad, lo cual trae como consecuencia la existencia de esquemas que pueden ser interpretados de diferentes maneras.

Por ejemplo considérese el siguiente esquema E-R:



A partir de la definición dada por Chen, el límite (Ø,1) de

la cardinalidad del conjunto entidad PROFESOR en APROBO, podría interpretarse de dos maneras:

a) Como una función parcial f1 tal que,

f1: ESTUDIANTE x MATERIA --> PROFESOR.

Así cada (est,mat) para el cual no esté definido f1 puede cumplir cualquiera de las siguientes condiciones:

a1) f1 no está definido para (est,mat) porque est no aprobó mat, razón por la cual, el profesor con el cual est aprobó mat es indeterminado.

a2) f1 no está definido para (est,mat) porque est aprobó mat por libre escolaridad, razón por la cual, no existe en PROFESOR un profesor con el cual haya aprobado la materia.

Las desventajas de interpretar la cardinalidad de esta forma es que en primer lugar no hay forma de diferenciar cual de las situaciones se está representando y, en segundo lugar, a menos que todos los elementos del producto cartesiano tengan sentido dentro de la relación, el límite inferior será cero.

b) Como una función f1 total tal que,

f1: EST_MAT --> PROFESOR U {NULO} , donde

EST_MAT={(est,mat)/est ∈ ESTUDIANTE, mat ∈ MATERIA y
est aprobó mat }

Siguiendo esta interpretación, el límite inferior cero se interpreta como que existen elementos (est,mat) que a pesar de cumplir que "est aprobó mat" no tienen asociado un elemento en PROFESOR.

De esta forma, los límites de la cardinalidad de un conjunto de objetos On en un conjunto relación R que relaciona los conjuntos de objetos O1,..,On se interpretan como el número

mimimo y máximo de ocurrencias de On con los cuales se pueden relacionar una ocurrencia de O1x..xOn-1 que tenga sentido dentro de R.

A lo largo de este artículo se seguirá la interpretación "b" de la cardinalidad.

Ambiguedades como ésta crean la necesidad de dar una definición formal del modelo, la cual pueda ser utilizada como base en la definición de la teoría de normalización.

Para dar esta especificación se ha utilizado el Lenguaje de la Lógica de Predicados y conceptos de la Teoría de Conjuntos. La misma está dividida en un conjunto de axiomas mediante los cuales se define el significado y las equivalencias de cada una de las abstracciones y, un conjunto de restricciones a través de las cuales, se presentan las condiciones que deben cumplirse cada vez que una abstracción es definida.

Así si se denota con ECi el esquema por comprensión de BDi, AXIOMAS(ECi) el conjunto de las fórmulas lógicas que expresan el significado de cada una de las estructuras de ECi, EE(ECi) el esquema por extensión de ECi, RI las restricciones de integridad entonces

$\forall$ ECi AXIOMAS(ECi) $\cup$ EE(ECi) $\cup$ ECi $\models$ RI , es decir, todas las estructuras de interpretación que sean modelos del conjunto de fórmulas AXIOMAS(ECi) $\cup$ EE(ECi) $\cup$ ECi son modelos también del conjunto de restricciones de integridad [1].

El modelo E-R utilizado es una modificación del modelo original definido por Chen al cual para aumentar su poder expresivo se han agregado: conjuntos de relaciones que relacionan

conjuntos de entidades y/o conjuntos de relaciones. Dependencias funcionales entre los atributos de un objeto. Límite inferior en la cardinalidad con la que participa un conjunto objeto en un conjunto relación.

Las abstracciones ofrecidas por el modelo E-R a utilizar son:

Conjunto de Entidades: grupo de entes que tienen propiedades en común. En la axiomatización se denotará un conjunto entidad cuyo nombre es **E** a través del siguiente predicado: nombre-conjunto-entidad(E).

Entidad: instancia u ocurrencia de un conjunto entidad. Si **E** es el nombre de un conjunto entidad y **e** es una instancia de **E** entonces entidad(e,E).

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 de relaciones con nombres **O1,..,On**, entonces nombre-conjunto-relacion(R,O1,..,On).

Relación: instancia u ocurrencia de un conjunto relación. Si (o1,..,on) es una instancia del conjunto relación R que relaciona los conjuntos entidad o relación **O1,..,On** de los cuales **o1,..,on** son instancias respectivamente, entonces se cumple el siguiente predicado: relacion(R(o1,..,on)).

Conjunto Valor: conjunto de valores que puede tomar una característica de un conjunto entidad o relación. Si**V** es el nombre de un conjunto valor entonces nombre-conjunto-valor(V).

Valor: instancia u ocurrencia de un conjunto valor. Si v es una instancia del conjunto valor V entonces V(v).

Atributo: es una característica de un conjunto entidad o

relación. Si Atr es un atributo del conjunto entidad o relación O que puede tomar valores del conjunto de valores V entonces atribucion(Atr,O,V).

Objeto: entidad o relación. Si o es una instancia del conjunto entidad o relación O entonces objeto(o,O).

Conjunto Objeto: conjunto entidad o relación. Si O es un conjunto entidad o relación entonces nombre-conjunto-objeto(O).

Todas estas abstracciones cumplen los siguientes axiomas:

```
objeto(e,E) <= entidad(e,E).

objeto(R(o1,..,on),R) <= relacion(R(o1,..,on)).

nombre-conjunto-objeto(E) <= nombre-conjunto-entidad(E).

nombre-conjunto-objeto(R) <= nombre-conjunto-objeto(O1) &...&
                             nombre-conjunto-objeto(On) &
                             nombre-conjunto-relacion(R,O1,..,On))

nombre-conjunto-entidad(E) <= entidad(e,E).

nombre-conjunto-relacion(R,O1,..,On) <= objeto(o1,O1)&..&
                                        objeto(on,On) &
                                        relacion(R(o1,..,on)).
nombre-conjunto-valor(V) <= V=entero v V=natural v V=caracter v
                            V=cad-dig v V=cad-car.

nombre-conjunto-valor(V1x..xVn) <= nombre-conjunto-valor(V1)&..&
                                   nombre-conjunto-valor(Vn).
```

Por ejemplo cada vez que se defina At como un atributo de un conjunto de objeto O sobre el conjunto de valores V se debe satisfacer la siguiente restricción de integridad:

```
∀o∀O∀At∀V(nombre-conjunto-objeto(O) & objeto(o,O) &
                    nombre-conjunto-valor(V) & atribucion(At,O,V)=>
                    V(At(o)) v At(o)=NULO ).
```

Por otro lado, además de estas abstracciones, el modelo permite representar las siguientes restricciones:

De cardinalidad de un conjunto de objetos O en un conjunto relación R. Así si On es el nombre de un conjunto de objetos, R /

es el nombre de un conjunto relación que relaciona a los conjuntos de objetos O1,..,On y la cardinalidad de On en R es min, max entonces cardinalidad(R,On,(min,max)).

De dependencia funcional entre los atributos de un conjunto de objetos. Si los atributos At1,..,Atn implican funcionalmente al atributos At en el conjunto de objetos O entonces dep-fun(((At1,..,Atn),At),O).

Luego cada vez que cardinalidad(R,On,(min,max)) se cumpla, se debe satisfacer la siguiente restrición de integridad:

Si o1,..,on-1 son elementos del dominio de On en R, es decir, pertenece al conjunto de elementos de O1x..xOn-1 que tienen sentido dentro de R, entonces o1,..,on-1 se relaciona a través de R como mínimo con **min** elementos de On y como máximo con **max**.

cardinalidad(R,On,(min.max)) <=>

```
∀o1..∀on-1(objeto(o1,O1) &...& objeto(on-1,On-1) &
           nombre-dominio(R1,R,On) & relacion(R1(o1,..,on-1)) =>
           min = < |{ (o1,..,on)|relacion(R(o1,..,on))}| =< max )
```

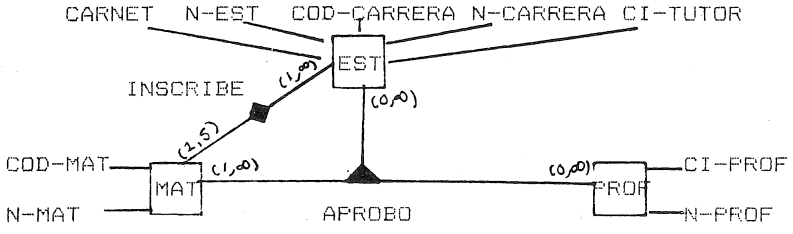
El dominio de un conjunto de objetos On en el conjunto relación R es R1 si, R1 es el conjunto de todos los elementos de O1x..xOn-1 que están relacionados con algún elemento de On o con el valor NULO. El valor NULO se interpreta como un elemento que no pertenece a On y es diferente al indeterminado y al desconocido.

nombre-dominio(R1,R,On) <=>

```
∀o1..∀on-1(relacion(R1(o1,..,on-1)) =>
∃on( (objeto(on,On) & relacion(R(o1,...,on))) ∨
       ( nombre-conjunto-relacion(R,O1,..,On ∪ {NULO}) =>
         relacion(R(o1,...,NULO)))).
```

4. FORMAS NORMALES MODELO ENTIDAD RELACION

Considérese el siguiente esquema E-R que representa la información de un estudiante, la carrera que cursa, las materias que inscribe, las materias que ha aprobado con un profesor y el profesor que es su tutor.



En el conjunto entidad EST, CARNET es la clave, además, existe la dependencia funcional COD-CARRERA --> N-CARRERA.

En el conjunto relación APROBO existe la dependencia funcional EST,MAT --> PROF.

Considérense las siguientes operaciones sobre el esquema:

a) Modificar el nombre de la carrera Física Aplicada por Ingeniería Física. Para mantener la consistencia, se deberían actualizar todas las entidades de todos los estudiantes que cursen Física Aplica. Si queda alguno sin actualizar, se genera una inconsistencia.

b) Agregar la carrera Ingeniería Geofísica la cual no estudia algún estudiante. Dado que no puede existir un estudiante sin valor en su CARNET, no se puede agregar mientras no exista un estudiante en la CARRERA.

c) Eliminar el único estudiante que pertenecía a la carrera Ingeniería de Suelos. Originando con esto que se pierda la información de la carrera.

d) Agregar al estudiante Luis Perez el tutor cuya CI-TUTOR

es 5313888. Qué ocurre si el tutor no pertenece al staff de profesores?. Se está creando una inconsistencia.

e) Relacionar a través de APROBO al estudiante est1 con la materia mat1 y el profesor prof1 y, al estudiante est1 con la materia mat2 y el profesor prof2. Dado que (est,mat) se relaciona a través de APROBO con prof1 y prof2, se viola la dependencia funcional EST,MAT-->PROF.

Estos problemas se presentan en el esquema porque en el conjunto entidad EST se mantienen dependencias funcionales que no se expresan explícitamente y se hace referencia a los elementos de un conjunto entidad PROF y no se establece explícitamente las restricción de integridad referencial. Además entre los elementos del conjunto relación APROBO se mantiene una dependencia funcional que tampoco se expresa explícitamente.

Para evitar estos tipos de problemas se definen las siguientes formas normales para el modelo E-R.

Un esquema E-R está normalizado si todo conjunto entidad y conjunto relación del esquema está normalizado.

nombre-esquema(E1) & normalizado(E1) <=>

∀E∀R(nombre-conjunto-entidad(E) & nombre-conjunto-relacion(R) &
conjunto-entidad(E,E1) & conjunto-relacion(R,E1) =>
normalizado(E) & normalizado(R)).

Sea E un conjunto entidad se dice que E está normalizado si todo atributo depende únicamente de las claves de E y no existe un atributo Atr en E tal que exista otra entidad tal que At sea clave o parte de la clave.

nombre-conjunto-entidad(E) & normalizado(E) <=>

∀la∀At(clave(la,E) & atribucion(At,E,V) => dep-fun(la,At,E) &
¬∃la1(¬clave(la1,E) & dep-fun(la1,At,E))) &
¬∃Atr(atribucion(Atr,E,V) & ∃E1(nombre-conjunto-entidad(E1) &
clave((At1,,,.Atr),E1))).

Cumpliendo esta forma normal, el conjunto entidad se debería descomponer en dos conjuntos entidad EST y CARRERA y, en dos conjuntos relación ES-TUTOR y ESTUDIA. El conjunto entidad EST con atributos CARNET, N-EST. El conjunto entidad CARRERA con atributos COD-CARRERA, N-CARRERA. El conjunto relación ES-TUTOR relaciona EST con PROF y, cardinalidad(ES-TUTOR,EST,(0,∞)) y cardinalidad(ES-TUTOR,PROF,(0,1)). El conjunto relación ESTUDIA relaciona EST con CARRERA y, cardinalidad(ESTUDIA,CARRERA,(0,1)) y cardinalidad(ESTUDIA,EST,(0,∞)).

En esta descomposición los problemas que se presentaban anteriormente en el conjunto entidad EST ya no se presentan.

Falta ahora solucionar los problemas que se presentan en el conjunto entidad APROBO.

Sea R un conjunto relación, se dice que R está normalizado si no existe un subconjunto S de la clave tal que exista una dependencia funcional X que se mantenga en R y S sea implicante de X , todo elemento que no pertenezca a una clave o sea atributo de R debe depender únicamente de las claves de R y, no existe un atributo Atr en R tal que exista otra entidad tal que At sea clave o parte de la clave.

nombre-conjunto-relacion(R,O1,..,On) & normalizado(R) <=>

∀la(clave(la,R)&¬∃la1(sub-con-propio(la1,la)&implicante(la1,R))) &
∀at(elemento(at,R) => ¬∃la(¬clave(la,R) & de-fun((la,at),R))) &
¬∃Atr(atribucion(Atr,R,V) & ∃E1(nombre-conjunto-entidad(E1) &
clave((At1,,,.Atr),E1))).

Sea X una dependencia funcional que se mantiene en el objeto

O de la forma S -> P, donde S denota un grupo de atributos y P denota un atributo entonces, S es un implicante en X en el objeto O.

implicante(l1,O) <=> ∃At(dep-fun((l1,At),O)).

Cumpliendo esta forma normal, en el conjunto relación APROBO se debe representar a través de las restricciones de cardinalidad, las dependencia funcional EST,MAT-->PROF. Eso se logra modificando la cardinalidad de PROF en APROBO es decir, cardinalidad(APROBO,PROF,(0,1)).

Es importante destacar la siguiente propiedad de los esquemas E-R normalizados:

PROP. Sea E1 un esquema E-R normalizado, sea E2 un esquema Relacional tal que E2=TRADUCCION(E1), donde TRADUCCION es el algoritmo de traducción de Markowitz[15], entonces E2 cumple como mínimo la 3FN.

5. TECNICAS DE NORMALIZACION DE ESQUEMAS E-R.

En el trabajo realizado, además de definir formas normales para el modelo E-R, se han establecido criterios de equivalencia y se han definido algoritmos de traducción de esquemas no normalizados a esquemas normalizados.

El mejor de estos algoritmos debido a que preserva toda la información semántica que se representaba originalmente, es el algoritmo que se basa en transformar la fórmula lógica que representa el esquema E-R en un fórmula equivalente que representa un esquema E-R normalizado, haciendo sustituciones sucesivas. Los pasos de este algoritmo son:

1) Traducir el esquema E-R a la conjunción de las fórmulas

lógicas del conjunto de axiomas y restricciones de integridad.

2) Sustituir cada subfórmula correspondiente a un conjunto de objetos con dependencias funcionales redundantes, por una subfórmula equivalente que corresponde al mismo conjunto de objetos pero sin las dependencias funcionales redundantes.

3) Sustituir cada subfórmula correspondiente a un conjunto entidad no normalizado, por una fórmula equivalente, que corresponde a un esquema E-R normalizado.

4) Sustituir cada subfórmula correspondiente a un conjunto relación no normalizado, por una fórmula equivalente, que corresponde a un esquema E-R normalizado.

5) Traducir la fórmula normalizada, al esquema E-R correspondiente.

Con este algoritmo se logra hacer un proceso de normalización a nivel del modelo E-R, que preserva la información que originalmente se quería representar.

6. TECNICAS DE TRADUCCION DE E-R A RELACIONAL.

Como se dijo anteriormente, no existen manejadores de bases de datos comerciales cuya herramienta de especificación es el modelo E-R. Luego es de interés la existencia de algoritmos de traducción a los modelos que si ofrecen los manejadores que tengan la propiedad de preservar en el esquema resultado, toda la semántica que originalmente se había representado.

Existen varios algoritmos de traducción de esquemas E-R a Relacional, por ejemplo el de Ullman, y el Markowitz[15,20], entre otros. En ambos existe pérdida semántica al hacer la traducción a relacional, ya que por ejemplo con el de Markowitz no se representa el límite inferior de la cardinalidad ni

tampoco, el límite superior cuando es diferente de infinito. Por otro lado, utilizando el algoritmo de Ullman, además de perder esta información, no se representa la información que depende del rol que juegan los objetos en una relación.

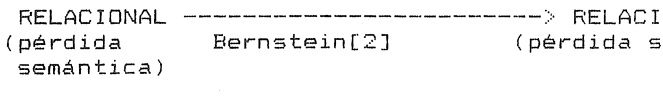
Para subsanar estas limitaciones, en el trabajo se propone un algoritmo que traduce todas las abstracciones que no se pueden representar a través de una estructura del relacional, como una restricción de integridad. Por ejemplo: la restricción que un estudiante no puede inscribir menos de 2 materias ni más de 5, que en el E-R se puede representar a través de cardinalidad(INSCRIBE,MAT,(2,5)), representarla en relacional de la siguiente manera:

$$\neg\exists\ est\ (\ |\{\ (est,mat)|\ tupla((est,mat),INSCRIBE)\ \}|\ <\ 2\ o$$
$$5\ <\ |\{\ (est,mat)|\ tupla((est,mat),INSCRIBE)\ \}|$$

Donde INSCRIBE es una relación del esquema relacional y tupla(x,R) sí "x es una instancia de la relación R".

CONCLUSIONES.

Tradicionalmente para poder definir una base de datos normalizada se tenían que seguir los siguientes pasos:

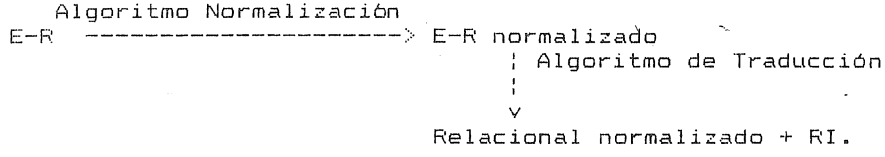


Aparte de la pérdida de información que se presentaba en cada nivel de traducción y que el diseñador conceptual no siempre podía verificar si el esquema normalizado realmente correspondía con la realidad que quería representar, se presentaban problemas

cuando no se utilizaba el modelo Relacional para implementar la BD.

Ofreciendo un Teoría de Normalización para un modelo de Datos más semántico y definiendo algoritmos apropiados de traducción, se pueden subsanar las limitaciones anteriormente presentadas.

De esta forma, para el caso en que sí se implementa la B.D con el modelo Relacional y se realiza la modelación conceptual con E-R, se puede obtener un esquema normalizado que realmente modela toda la información que originalmente se había representado. Así, se podría ver el proceso de diseño de una Base de Datos de la siguiente forma:



Además estos algoritmos pueden ser utilizados para herramientas CASE que asistan en el diseño de BD. De hecho los mismos serán utilizados en el modulo de diseño de BD de la herramienta CASE que se está desarrollando en la Universidad Simón Bolívar.

Una posible extensión a este trabajo es extender la teoría al modelo de datos Entidad Relación Extendido[18].

BIBLIOGRAFIA.

0. BAGCHI, T. y CHAUDHRI V. "Interactive Relational Database Design". Springer-Verlag. 1987.

1. BARRY, E. J. "Applied Database Logic", Vol I Fundamental Database Issues. Prentince-Hall, 1985.

2. BERNSTEIN, P. A. "Synthesizing third normal form relations from

functional dependencies". ACM Trans. Database Syst. 1, 4 (Dec.1976).

3. BERRI, C. y BERSTEIN, P. A. "Computacional problems related to the design of normal form relational schemas". ACM Trans. Database Syst. 4, 1, (Mar.1979).

4. CERI, S. y GOTTLOB, G. "Normalization of Relations and Prolog", Communications of the ACM. 29, 6, (Jun,1986).

5. CODD, E. F. "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM. 13, 6, (Jun,1970).

6. CODD, E. F. "Extending the Database Relational Model to Capture More Meaning", ACM Trans. Database Syst. 4, 4, (Dec 1979).

7. CHEN, P.P. "The Entity_Relationship Model-Toward a Unified View of Data", ACM Trans. Database Syst. 1, 1, (Mar 1976).

8. DATE, C. J. "An Introduction to Database Systems", Fourth Edition 1986. Addison-Wesley.

9. DIEDERICH, J. y MILTON, J. "New Methods and Fast Algoritms for Database Normalization". ACM Trans. Database Syst. 13, 3,(Sep 1988).

10. FAGIN, R. " Multivalued dependencies and new form for relational database", ACM Trans. Database Syst. 2, 3 (Sep 1977).

11. FAGIN, R., MENDELZON, A., ULLMAN J. "A Simplified Universal Relation Assuption and its Properties", ACM Trans. Database Syst. 7,3 (Sep 1982).

12. HULL R. y KING R., "Semantic Database Modeling: Survey, Applications, and Research Issues", ACM Comp.Surv. 19, 3 (Sep 1987).

13. MAEIER, D. "Minimun Covers in the Relational Database Model" J. ACM, 27, 4 (Oct 1980).

14. MAEIER, D., SAGIV Y., YANNAKAKIS M. "On the Complexity of Testing Implications of Functional and Join Dependencies" J. ACM, 28, 4 (Oct 1981).

15. MARKOWITZ V. y SHOSHANI A. "On the correctness of representing Extended Entity_Relationship structures in the Relational Model", ACM Procc. 1989.

16. PECKHAM J. y MARYANSKI F. "Semantic Data Models" ACM Comp. Surv. 20, 3 (Sep 1988).

17. SILVEIRA P. "Procedural Data Manipulation Operations for E-R Model" Conferencia Latinoamericana de Informática 1990.

18. TEORY T., YANG D., FRY J. "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model". ACM Comp. Surv. 18, 2 (Jun 1986).

19.TSICHRITZIS, D. y LOCHOVSKY, F. "Data Models". 1982.

20. ULLMAN J. "Principles of Database and Knowledge-Base Systems", Vol I. 1988. Computer Science Press.

First-Order Liveness Analysis via Type Inference

—DRAFT—

Juan Carlos Guzmán* Paul Hudak*

March 15, 1991

Abstract

In this paper we describe how to perform *liveness analysis* as a by-product of type inference in an extended Hindley-Milner type system. The analysis captures an approximation to the lifetime properties of objects, including polymorphic and recursive datatypes.

Our system relies on nested quantification over "liveness labels", which contrasts sharply with the Hindley-Milner system, in which quantification is restricted to the outermost level. Thus we cannot rely on a simple unification strategy to compute liveness of recursive functions and data structures. The system does have a "most-general-type" property, and we present an effective algorithm that computes such a type.

[1]Authors' address: Yale University
Department of Computer Science
P.O. Box 2158 Yale Station
New Haven, CT 06520
{guzman-juan,hudak-paul}@cs.yale.edu

1 Introduction

There has been considerable interest in recent years in using static type systems to specify constraints on and infer properties of program behaviors other than the normal notion of "type." Examples include Gifford, Lucassen, and Jouvelot's various uses of *effect systems* to capture mutability properties [LG88, GL86, JG89], Kuo and Mishra's type system that performs *strictness analysis* [KM89], Baker's simple extension to the Hindley-Milner type system to capture certain sharing properties [Bak90], and our own work on *single-threaded polymorphic lambda calculus* [GH90].

There are several motivations for using a static type system for these purposes instead of, for example, an abstract interpretation specification. First, the notation has certain advantages. In particular, the inference rules are specified in a modular way, with the inferencing algorithm specified independently. In contrast, denotational specifications generally rely on an understanding of domain theory, and the algorithm for inferring the properties is intrinsically tied to the inference rules.[1]

Second, we are interested not just in *inferring* program properties, but also in specifying certain *constraints*, or well-formedness criteria, on programs. A type system is the conventional method for specifying such static, context sensitive, constraints, and so is a natural choice. In addition, it allows the user to specify contraints explicitly by providing "type" signatures for functions, modules, etc.

Finally, for strongly-typed languages, the conventional type structure can be used as a "carrier" for whatever additional properties are being specified or inferred. In this context types have the following nice properties:

- They represent a *partition* of the domain of objects.
- They are *finite*.
- They reflect accurately the *structure* of objects (e.g., non-atomic and recursive types).

Thus properties that rely on notions of type structure can be specified as extensions to the conventional type structure, which simplifies the overall specification and aids reasoning about program correctness.

In this paper we describe how to perform *liveness analysis* as a by-product of type inference in an extended Hindley-Milner type system. The analysis captures an approximation to the lifetime properties of objects, including polymorphic and recursive datatypes. Although such information is of obvious use in a compiler (allowing, for example, certain forms of compile-time garbage collection), we also see its manifestation in a type system as a vehicle for designing languages whose static semantics depends upon some notion of object liveness.

[1] There has been some speculation that any abstract interpretation can be described within a type inference framework, but no general result of this sort is known.

For example, a language with "single-threaded" or "linear" types needs some notion of liveness on which to base the linearity (indeed, our result is a significant improvement over the liveness captured in our previous work [GH90]).

As an example of what our system is able to infer, consider the following function definitions:

$$\begin{array}{lcl} \texttt{f(x,y)} & = & \textit{if } x = y \textit{ then } x \textit{ else } x \\ \texttt{g(x,y)} & = & \textit{if } x = y \textit{ then } y \textit{ else } y \\ \texttt{h(x,y)} & = & \textit{if } x = y \textit{ then } x \textit{ else } y \end{array}$$

The conventional first-order Hindley-Milner type signature for each of these functions is the same (we assume the domain product operator binds tighter than the function operator):

$$\texttt{f,g,h} : \forall\alpha.\alpha \times \alpha \rightarrow \alpha$$

However, note that f discards its first argument and returns the second, g does the opposite, and h does one or the other depending on the equality of the two arguments. Thus our type system infers the following "types:"

$$\begin{array}{ll} \texttt{f} :: & \forall\alpha.\alpha^{\theta_1} \times \alpha^{\theta_2} \rightarrow \alpha^{\theta_1} \\ \texttt{g} :: & \forall\alpha.\alpha^{\theta_1} \times \alpha^{\theta_2} \rightarrow \alpha^{\theta_2} \\ \texttt{h} :: & \forall\alpha.\alpha^{\theta_1} \times \alpha^{\theta_2} \rightarrow \alpha^{\theta_1,\theta_2} \end{array}$$

The superscript on a type captures the liveness property. The way to read the liveness of f, for example, is: "f's first argument may persist as a result of applying f" or, alternatively, "f's result may contain references to its first argument." Similarly, for h: "h's result may contain references to either of its arguments."

As another example, the type of cons is:

$$\texttt{cons} : \forall\alpha.\alpha^{\theta_1} \times (\mathit{List}^{\theta_2}\ \alpha^{\theta_3}) \rightarrow (\mathit{List}^{\theta_2}\ \alpha^{\theta_1,\theta_3})$$

Note here that the type constructor *List* is annotated as well. We can interpret the above as: "The result returned from cons is a list which may reference part of the list structure of the second argument, and may contain elements that reference part of the first argument or elements of the second argument."

Aside from the obvious pragmatic advantages of our system, we have worked out some interesting theoretical difficulties relating to the type inference (i.e. reconstruction) problem. In particular, our system relies on nested quantification over liveness labels, which contrasts sharply with the Hindley-Milner system, in which quantification is restricted to the outermost level. Thus we cannot rely on a simple unification strategy to compute liveness of recursive functions and data structures. The system does have a "most-general-type" property, and we present an effective algorithm that computes such a type.

2 Liveness Analysis

We begin by defining the syntax of a simple functional language that will be used for all of our examples:

f	$\in$	$FIde$	function identifiers
v, x	$\in$	$VIde$	value identifiers
		$Ide = FIde + VIde$	identifiers
k	$\in$	Kon	constants
k_v	$\in$	Kon_v	constant values
k_f	$\in$	Kon_f	constants functions
e	$\in$	Exp	expressions
g	$\in$	$Funct$	functions

$$
\begin{array}{rcl}
\text{where } g & ::= & f \mid k_f \\
e & ::= & k_v \mid v \\
& & \mid g(e_1, \ldots, e_n) \\
& & \mid \mathit{let}\ v_1 = e_1; \\
& & \quad \ldots; \\
& & \quad v_m = e_m; \\
& & \quad f_1(x_{11}, \ldots, x_{1q_1}) = e'_1; \\
& & \quad \ldots; \\
& & \quad f_n(x_{n1}, \ldots, x_{nq_n}) = e'_n \\
& & \mathit{in}\ e
\end{array}
$$

Next we specify the syntax of type schemes, which is essentially that of Hindley-Milner [Hin78, Mil78, DM82], restricted to the first-order case:

$\sigma^v \in TypeSch$	::=	τ	Value Type Scheme
		$\mid \forall\alpha.\sigma^v$	Universal Quantification
$\sigma^f \in TypeSch$	::=	ϕ	Functional Type Scheme
		$\mid \forall\alpha.\sigma^f$	Universal Quantification
$\tau \in Type^v$	::=	$Int \mid Bool \mid \ldots$	Basic Types
		$\mid Pair\ \tau_1\ \tau_2$	Pairs
		$\mid List\ \tau$	Lists
		$\mid \alpha$	Variables
$\phi \in Type^f$	::=	$\mid \tau_1 \times \ldots \times \tau_n \rightarrow \tau_0$	Functions

We also assume the presence of the constants shown in Figure 1, having their traditional semantics.

$$
\begin{aligned}
\texttt{if}: &\ \forall\alpha.\mathit{Bool} \times \alpha \times \alpha \rightarrow \alpha \\
\texttt{null?}: &\ \forall\alpha.(\mathit{List}\ \alpha) \rightarrow \mathit{Bool} \\
\texttt{nil}: &\ \forall\alpha.(\mathit{List}\ \alpha) \\
\texttt{cons}: &\ \forall\alpha.\alpha \times (\mathit{List}\ \alpha) \rightarrow (\mathit{List}\ \alpha) \\
\texttt{head}: &\ \forall\alpha.(\mathit{List}\ \alpha) \rightarrow \alpha \\
\texttt{tail}: &\ \forall\alpha.(\mathit{List}\ \alpha) \rightarrow (\mathit{List}\ \alpha) \\
\texttt{pair}: &\ \forall\alpha\beta.\alpha \times \beta \rightarrow (\mathit{Pair}\ \alpha\ \beta) \\
\texttt{fst}: &\ \forall\alpha\beta.(\mathit{Pair}\ \alpha\ \beta) \rightarrow \alpha \\
\texttt{snd}: &\ \forall\alpha\beta.(\mathit{Pair}\ \alpha\ \beta) \rightarrow \beta
\end{aligned}
$$

Figure 1: Standard Type Signatures for Selected Constants

3 Syntax of Extended Types

In this section we extend our simple language with syntax to support liveness properties. Although verbose upon initial inspection, we will soon adopt some conventions to simplify the notation.

$\hat{\sigma} \in \mathit{TypeSch}_{Ext} ::=$	$\hat{\sigma}^v \mid \hat{\sigma}^f$	Extended Type Schemes
	$\hat{\chi}^v$	Extended Value Type Scheme
	$\mid \forall\alpha.\hat{\sigma}^v$	
$\hat{\sigma}^v \in \mathit{TypeSch}^v_{Ext} ::=$	$\hat{\chi}^v$	Extended Value Type Scheme
	$\mid \forall\alpha.\hat{\sigma}^v$	
$\hat{\chi}^v \in \mathit{LivenessSch}^v_{Ext} ::=$	$\hat{\tau}$	
	$\forall\theta.\hat{\chi}^v$	
$\hat{\sigma}^f \in \mathit{TypeSch}^f_{Ext} ::=$	$\hat{\chi}^f$	Extended Functional Type Scheme
	$\mid \forall\alpha.\hat{\sigma}^f$	
$\hat{\chi}^f \in \mathit{LivenessSch}^f_{Ext} ::=$	$\hat{\phi}$	
	$\forall\theta.\hat{\chi}^f$	
$\hat{\tau} \in \mathit{ObjType}_{Ext} ::=$	$\mathit{Int}^\delta \mid \mathit{Bool}^\delta \mid \ldots$	Basic Types
	$\mid (\mathit{Pair}\ \hat{\tau}_1\ \hat{\tau}_2)^\delta$	Pairs
	$\mid (\mathit{List}\ \hat{\tau})^\delta$	Lists
	$\mid \alpha^\delta$	Variables
$\hat{\phi} \in \mathit{FuncType}_{Ext} ::=$	$\hat{\tau}_1 \times \ldots \times \hat{\tau}_n \rightarrow \hat{\chi}^v$	Function Arguments ($n \geq 1$)
	$\mid \forall\theta.\hat{\phi}$	
$\delta \in \mathit{LivenessExp} ::=$	$\theta_1, \ldots, \theta_n$	Liveness Label Sequence ($n \geq 1$)

As should be clear from the examples given in the introduction, liveness information is expressed by annotating each type expression with a "liveness expression" ($\mathit{LivenessExp}$). The θ's are called *liveness labels*, and should not be confused with *type variables*; indeed,

they are not variables at all. The $\forall$ quantifier binds liveness labels, and is only permitted at the functional type level (informally, the liveness information of the argument is 'introduced' by universal quantification), and in the outermost level of the range of a function. The latter kind of quantification is found, for example, in the function `pair` which creates a new pair each time it is called.

Intuitively, liveness labels model the different "regions," or "pools" where objects can "live." A region can only host objects of one type. If the type is atomic (as in the case of Int^{θ}) objects are in the region (θ in this case). If the type is structured, like $(Pair\ Int^{\theta_1}\ Bool^{\theta_2})^{\theta_3}$, then only the "skeleton" of the type is in the region (in this case the pair cell), but the objects are not (in this case, the integer is in region θ_2, and the boolean in θ_3). Since regions are monomophic, θ_1, θ_2, and θ_3 must all be different. A type labeled with several labels, as in Int^{θ_1,θ_2}, indicates that the object may belong to either region.

A polymorphic functional type, such as $\alpha^{\theta} \rightarrow \alpha^{\theta}$, must expect its argument to be from any region, and thus we quantify over the labels of its arguments, as in:

$$\forall\theta.\alpha^{\theta} \rightarrow \alpha^{\theta}$$

with the meaning that the 'region' the result belongs to is the same as the argument's. A type such as:

$$\forall\theta_1.\alpha^{\theta_1} \rightarrow \forall\theta_2.\alpha^{\theta_2}$$

means that the result will be in an unspecified region, which cannot be assumed to be the same as the argument's. This is why we need nested quantification.

From the above specification of extended types, the functions `f`, `g`, and `h` shown in the introduction have the following types:

$$\begin{aligned}
\texttt{f} &:: \forall\alpha.\forall\theta_1\theta_2.\alpha^{\theta_1} \times \alpha^{\theta_2} \rightarrow \alpha^{\theta_1} \\
\texttt{g} &:: \forall\alpha.\forall\theta_1\theta_2.\alpha^{\theta_1} \times \alpha^{\theta_2} \rightarrow \alpha^{\theta_2} \\
\texttt{h} &:: \forall\alpha.\forall\theta_1\theta_2.\alpha^{\theta_1} \times \alpha^{\theta_2} \rightarrow \alpha^{\theta_1\theta_2}
\end{aligned}$$

3.1 Syntactic Conventions

It is quite common to have many different liveness labels within a type expression – technically, each and every node is entitled to one – and each has an associated point at which it is quantified. But it provides useful information only when it appears in several places, thus capturing a relationship between those places. Therefore to simplify the notation we adopt the following conventions:

1. η will be used instead of θ when θ is bound in the innermost $\hat{\chi}$ (either $\hat{\chi}^v$, or $\hat{\chi}^f$) that encloses all ocurrences of θ, and the binding of η will be omitted. As a result, the type for the example function `f` given in the introduction will be written as:

   $$\texttt{f} :: \forall\alpha.\alpha^{\eta_1} \times \alpha^{\eta_2} \rightarrow \alpha^{\eta_1}$$

   Further, we omit liveness variables that:

$$
\begin{array}{rl}
\texttt{if}: & \forall\alpha.Bool \rightarrow \alpha^{\theta_1} \rightarrow \alpha^{\theta_2} \rightarrow \alpha^{\theta_1\,\theta_2} \\
\texttt{null?}: & \forall\alpha.(List\ \alpha) \rightarrow Bool \\
\texttt{nil}: & \forall\alpha.(List\ \alpha) \\
\texttt{cons}: & \forall\alpha.\alpha^{\theta_1} \rightarrow (List^{\theta_2}\ \alpha^{\theta_3}) \rightarrow (List^{\theta_2}\ \alpha^{\theta_1,\theta_3}) \\
\texttt{head}: & \forall\alpha.(List\ \alpha^{\theta}) \rightarrow \alpha^{\theta} \\
\texttt{tail}: & \forall\alpha.(List^{\theta_1}\ \alpha^{\theta_2}) \rightarrow (List^{\theta_1}\ \alpha^{\theta_2}) \\
\texttt{pair}: & \forall\alpha\beta.\alpha^{\theta_1} \rightarrow \beta^{\theta_2} \rightarrow (Pair\ \alpha^{\theta_1}\ \beta^{\theta_2}) \\
\texttt{fst}: & \forall\alpha\beta.(Pair\ \alpha^{\theta}\ \beta) \rightarrow \alpha^{\theta} \\
\texttt{snd}: & \forall\alpha\beta.(Pair\ \alpha\ \beta^{\theta}) \rightarrow \alpha^{\theta}
\end{array}
$$

Figure 2: Extended Types for Constants

(a) are bound according to this convention, and

(b) are used only once in the type expression, and it is the only label annotating the type node (i.e., is not part of a l.u.b.).

The type of `f` is then further simplified to

$$\texttt{f} :: \forall\alpha.\alpha^{\eta_1} \times \alpha \rightarrow \alpha^{\eta_1}$$

2. We will write $(List^{\theta}\ \hat{\tau})$ and $(Pair^{\theta}\ \hat{\tau_1}\ \hat{\tau_2})$ instead of $(List\ \hat{\tau})^{\theta}$ and $(Pair\ \hat{\tau_1}\ \hat{\tau_2})^{\theta}$, respectively. Following this convention, the extended type of `tail` is written as

$$\texttt{tail}: \forall\alpha.(List^{\theta_1}\ \alpha^{\theta_2}) \times (List^{\theta_1}\ \alpha^{\theta_2})$$

3. We will write $\{\theta_1,\ldots,\theta_n\}\hat{\chi}$ when we need to make explicit that $\hat{\chi}$ has free liveness labels θ_1 through θ_n. Sometimes, we will write $\Theta\hat{\chi}$, if the particular variables are not important, but only the fact that some free labels exists.

4. We will write θ^{α} instead of θ when we need to stress the fact that label θ always qualifies the type variable α.

Following the conventions, the types for `f`, `g`, and `h` specified above are:

$$
\begin{array}{ll}
\texttt{f} :: & \forall\alpha.\alpha^{\eta_1} \times \alpha \rightarrow \alpha^{\eta_1} \\
\texttt{g} :: & \forall\alpha.\alpha \times \alpha^{\eta_2} \rightarrow \alpha^{\eta_2} \\
\texttt{h} :: & \forall\alpha.\alpha^{\eta_1} \times \alpha^{\eta_2} \rightarrow \alpha^{\eta_1,\eta_2}
\end{array}
$$

Figure 2 show the type signatures for the assumed constants, using the above conventions. Note that when all liveness information is removed, the resulting type is exactly the standard type for the constant.

3.2 Well-Formed Extended Types

In order to keep the distinction of regions to a minimun, we note that if a 'new' region is always associated with a pre-existing one, then it is not necessary to distinguish them. Consider cons, for example: it takes an object and a list, and creates a new cell containing the object and a pointer to the list. Its extended type could thus be:

$$\texttt{cons}: \forall\alpha.\alpha^{\eta_1} \times (List^{\eta_2}\ \alpha^{\eta_3}) \rightarrow (List^{\eta_2,\eta_4}\ \alpha^{\eta_1,\eta_3})$$

but the fact that η_4 is a 'new region', suggests that we can assume the region η_2 has yet another cell, thus simplifying the type somewhat to:

$$\texttt{cons}: \forall\alpha.\alpha^{\eta_1} \times (List^{\eta_2}\ \alpha^{\eta_3}) \rightarrow (List^{\eta_2}\ \alpha^{\eta_1,\eta_3})$$

Another characteristic of functional types is that given that its argument can belong to any region, and has no restrictions other than those implied by its type,

- all free labels of the argument must appear only once there (i.e., the argument must be *linear* on all free labels), and
- all free labels of the argument must be universally quantified in the function's type.

Well-formed extended types must conform to these restrictions, as well as the additional restriction that all instances of the same label must be attached to identical type schemes. In particular, if an instance of a label is attached to a type variable, then all its instances must be attached to the same type variable.

The following type is well-formed:

$$\forall\alpha.\alpha^{\eta_1} \times (List^{\eta_2}\ \alpha^{\eta_3}) \rightarrow (List^{\eta_2}\ \alpha^{\eta_1,\eta_3})$$

But the following are not:

$$\forall\alpha.\alpha^{\eta_1} \times (List^{\eta_2}\ \alpha^{\eta_1}) \rightarrow (List^{\eta_2}\ \alpha^{\eta_1})$$
$$\forall\alpha.\alpha^{\eta_1} \times (List^{\eta_2}\ \alpha^{\eta_3}) \rightarrow (List^{\eta_2}\ \alpha^{\eta_1,\eta_2,\eta_3})$$
$$\forall\alpha\beta.(List^{\eta}\ \alpha) \rightarrow (List^{\eta}\ \beta)$$

The first type is ill-formed because η_1 is used non-linearly in the argument (once in the first, and once in the second argument), the second one is ill-typed because η_2 labels a list node in one instance, and a variable in another, and the third one is not valid because η is labelling lists of possibly different type.

For the remainder of the paper, we will consider only well-formed extended types.

3.3 Operations on Extended Types

3.3.1 Substitution

Substitution on extended types is an extension of substitution on standard types. The major problem on the extended type language is that substitution on types implies some substitution on the liveness labels associated, so that the pairing between types and labels is maintained. The actual definitions will appear in the final paper in the appendix. In this section, we will provide only an intuitive explanation on how they work. Suppose we have the extended type for g (as in the introduction)

$$\hat{\tau} = \alpha^{\eta_1} \times \alpha^{\eta_2} \rightarrow \alpha^{\eta_1 \cup \eta_2}$$

and want to do the substitution (on types) (i.e., suppose the call g (Pair x y) is found, where x and y have type α)

$$\tau[(Pair\ \alpha_1\ \alpha_2)/\alpha]$$

This substitution imposes 'structural restrictions' on η_1, and η_2—these regions will now be subdivided into three parts, one for the Pair, skeleton, and one for each component of the pair. This implies that η_1, and η_2 need to be replaced by other labels, which will annotate the pairs. These labels need to be bound in the same place where the original ones were. The labels that replace η_1 must be different from those that replace η_2 (in order not to introduce additional constraints in the type), and α's that are annotated by both labels, need to be replaced by types that are annotated by the conjunction of the corresponding labels that replace η_1, and η_2—this can be achieved by *zipping* the replacements of α^{η_1}, and α^{η_2}. The following type results after the replacement:

$$\hat{\tau} = (Pair^{\eta_{11}}\ \alpha^{\eta_{12}}\ \alpha^{\eta_{13}}) \times (Pair^{\eta_{21}}\ \alpha^{\eta_{22}}\ \alpha^{\eta_{23}}) \rightarrow (Pair^{\eta_{11},\eta_{21}}\ \alpha^{\eta_{12},\eta_{22}}\ \alpha^{\eta_{13},\eta_{23}})$$

Substitution on liveness labels on the contrary is done in the standard way, provided the labels involved in the operation are *compatible* (they annotate the same type, or are newly introduced labels). E.g.,

$$(Pair^{\theta_1}\ \alpha^{\theta_2}\ \alpha^{\theta_3})[\theta_2/\theta_3] \equiv (Pair^{\theta_1}\ \alpha^{\theta_2}\ \alpha^{\theta_2})$$

3.3.2 Instance Relation

We are interested in building a partial order on types, using the substitution relation. Intuitively, one type is an instance of another if it can be obtained from the latter by applying a suitable substitution. Alternatively, it is also said that the latter type is 'more general' than the former one.

1. Parameter Type Instance

$$\hat{\sigma} \succeq \hat{\sigma}[\tau/\alpha]$$

2. Generic Type Instance

$$\forall\alpha.\hat{\chi} \succeq \forall\alpha'_1 \ldots \alpha'_n.\hat{\chi}[\hat{\tau}/\alpha]$$

when α_i are not free in $\forall\alpha.\hat{\chi}$.

3. Parameter Liveness Label Instance

$$\hat{\sigma} \succeq \hat{\sigma}[\delta/\theta]$$

4. Generic Liveness Label Instance on χ

$$\forall\hat{\theta}.\hat{\chi} \succeq \forall\theta_1 \ldots \theta_n.\hat{\chi}[\delta/\theta]$$

when θ_i are not free in $\forall\theta.\hat{\chi}$.

Relations 1, 2, and 3 are extended to work on all different categories of extended types ($\hat{\chi}$, and $\hat{\tau}$). Note that even though types are not shallow with respect to liveness labels, only outermost labels can be instantiated using rule 4. This is due to the lack of rules that would allow relating expresssions by using the relation of their constituents.

3.4 Extended-Type Inference Rules

The functions $\mathcal{K}^f$, and $\mathcal{K}^v$ return the extended type for constant functional, and value constants. Alternatively, these extended types can be encoded in the initial environment. An assertion of the form $T \vdash e : \hat{\chi}^v$ expresses the fact that under assumptions T, e has the extended type χ^v. The assertion $T \overset{f}{\vdash} g : \hat{\chi}^f$ expresses a similar fact but relating functions to their extended type. The inference rules are very similar to a typical Hindley-Milner type system, therefore we will restrict our comments on issues relevant to the liveness analysis.

Function Designators

1. Functional Constants

$$A \overset{f}{\vdash} k_f : (\mathcal{K}^f\ k_f)$$

The extended type of a functional constant looked up in $\mathcal{K}^f$.

2. Functional Identifiers

$$T \overset{f}{\vdash} f : A(f)$$

The extended type of a functional identifier is retrieved from the environment T.

Value Expressions

3. Value Constants

$$T \vdash k_v : (\mathcal{K}_v\ k_v)$$

4. Value Identifiers

$$T \vdash v : A(v)$$

5. Applications

$$\frac{\begin{array}{c} T \overset{f}{\vdash} g : (\forall\theta_{11}\ldots\theta_{1m_1}\ldots\theta_{n1}\ldots\theta_{nm_n}.\hat{\tau}_1 \times \ldots \times \hat{\tau}_n \to \hat{\chi}) \\ T \vdash e_1 : \hat{\tau}'_1 \quad \ldots \quad e_n : \hat{\tau}'_n \\ \hat{\tau}'_i = \hat{\tau}[\delta_{i1}/\theta_{i1}, \ldots, \delta_{im_i}/\theta_{im_i}] \qquad i = 1..n \end{array}}{T \vdash g(e_1, \ldots, e_n) : \hat{\chi}[\delta_{11}/\theta_{11}, \ldots, \delta_{nm_n}/\theta_{nm_n}]}$$

For technical reasons regarding well-formedness of extended-types, it is required that $\hat{\tau}'_i$ be a liveness-instance of $\hat{\tau}_i$. This is a departure from the Hindley-Milner inference rules, which require τ_i, and τ'_i to be the same.

1. Let expression

$$\frac{\begin{array}{c} T' = T\left[\begin{array}{c} v_1 \mapsto \hat{\chi}_1, \ldots, v_m \mapsto \hat{\chi}_m, \\ f_1 \mapsto \forall\theta_{11}\ldots\theta_{1p_1}.\hat{\tau}_{11} \times \ldots \times \hat{\tau}_{1q_1} \to \hat{\chi}'_1, \\ \ldots, \\ f_n \mapsto \forall\theta_{n1}\ldots\theta_{np_n}.\hat{\tau}_{n1} \times \ldots \times \hat{\tau}_{nq_n} \to \hat{\chi}'_n \end{array}\right] \\ T' \vdash e_1 : \hat{\chi}_1, \ldots, e_m : \hat{\chi}_m \\ T'[x_{11} \mapsto \hat{\tau}_{11}, \ldots, x_1 q_1 \mapsto \hat{\tau}_{1q_1}] \overset{f}{\vdash} e'_1 : \hat{\chi}'_1 \\ \ldots \\ T'[x_{n1} \mapsto \hat{\tau}_{n1}, \ldots, x_n q_n \mapsto \hat{\tau}_{nq_n}] \overset{f}{\vdash} e'_n : \hat{\chi}'_n \\ T' \vdash e : \hat{\tau} \end{array}}{T \vdash \left(\begin{array}{l} let\ v_1 = e_1; \\ \quad \ldots; \\ \quad v_m = e_m; \\ \quad f_1(x_{11}, \ldots, x_{1q_1}) = e'_1; \\ \quad \ldots; \\ \quad f_n(x_{n1}, \ldots, x_{nq_n}) = e'_n \\ in\ e \end{array}\right) : \hat{\tau}}$$

This rule, although complex in appearance, is a direct extension to the *let*-rule in the Hindley-Milner type system. Note that polymorphism is allowed in the type of the bound identifiers.

2. Generalization

$$\frac{\begin{array}{c} T \vdash e : \hat{\sigma} \\ \alpha \text{ not free in } T \end{array}}{T \vdash e : \forall\alpha.\hat{\sigma}}$$

$$\frac{T \vdash e : \hat{\chi} \quad \theta \text{ not free in } T}{T \vdash e : \forall\theta.\hat{\chi}}$$

3. Instantiation

$$\frac{T \vdash e : \hat{\chi} \quad \hat{\chi} \succeq \hat{\chi}'}{T \vdash e : \hat{\chi}'}$$

$$\frac{T \vdash e : \hat{\sigma} \quad \hat{\sigma} \succeq \hat{\sigma}'}{T \vdash e : \hat{\sigma}'}$$

Note that having the $\forall$ quantifier arbitrarily nested implies that expressions using this quantifier potentially generate a different instance every time the quantifier gets removed, i.e., every time the expression appears in the program text. This is reminiscent of the polymorphism introduced in let-expressions in a Hindley-Milner type system) where each time a let-bound value gets used, a 'fresh' instantiation of its type is manipulated.

4 Type Reconstruction

In this section we present how extended types can be effectively computed. Even though there exists algorithms that directly compute the 'extended type' of expressions, we have chosen to show an algorithm that merely 'extends the type' of a pre-inferred type. Such an algorithm has the following advantages:

- it works exclusively on liveness properties, and not on the basic type; this is a modular design in the same manner that lexer and parser are modular in compilation.
- No need for unification, just pattern matching, so, no need to to carry substitution environments around.

4.1 Overview

This algorithm departs from traditional type inferring algorithms in that this does not work 'top-down', but rather recognizes the dependencies among different identifiers in the program and always works on those identifiers for which all their dependencies have been solved. The following is an overview of the strategy used:

1. α-rename the program in such a way to avoid let-bound identifier name clashes.[2]

2. Build a *dependency graph* of the let-bound identifiers in the program as follows: if identifier x depends on identifier y for its value (a reference to y appears inside x's body), then put a dependence arc from x to y.

3. Collapse each group of nodes with cyclic dependencies in the graph into a *collection*, so that if x is dependent on y and vice versa, then x and y are both in the same collection. The resulting graph is a DAG.

4. Order the collections topologically, beginning with ones that have no outgoing edges (except, perhaps, to themselves).

5. Infer the type for nodes in collections in the order given by the topological sort. Each collection is to be solved using the technique given below.

4.1.1 Solving Each Collection

Identifiers that belong to the same collection are interdependent—the type of either of them may affect the value of the rest. Thus, we must solve all of them at the same time. Further, for collections bigger than one, the collection is recursive, thus we need to use fixpoint finding techniques to assign values to the different identifiers of the collection. We will do so by using a standard ascending Kleene chain technique.

The initial extended type of each element in the collection will be linear in all its liveness labels. E.g., if the type v is $(Pair\ Int\ Int)$, then its initial extended type will be $\forall\theta_1\theta_2\theta_3.Pair^{\theta_1}\ Int^{\theta_2}\ Int^{\theta_3}$; if f is of type $Int \rightarrow Int$, then its initial extended type will be $\forall\theta_1.Int^{\theta_1} \rightarrow \forall\theta_2.Int^{\theta_2}$. This represents the least extended type assignable to the identifier, since all other extended types are instances of this one.

Compute a new extended type for each identifier in the collection by assuming the current types and using algorithms $\mathcal{W}^f$, and $\mathcal{W}^v$. This is iterated until all the computed values match the assumed ones (modulo α-renaming).

4.2 Algorithm

Preliminaries

One of the benefits of the α-conversion performed in the first step of the algorithm is that we can associate to the program a *flat* environment containing the Hindley-Milner, and extended

[2]This is more a matter of convenience, since this way, a name uniquely identifies a program point, and thus also acts as a label.

types for every let-bound identifier. By abuse of notation we annotate the identifiers with their types rather than keeping an explicit environment.

$$\begin{aligned} T \in TEnv_{Ext} : &\ \ Ide \rightarrow TypeSch_{Ext} \\ e \in AExp : &\ \ Exp \times TEnv_{Ext} \end{aligned}$$

Given an annotated program, and a let-bound identifier, we obtain the definition corresponding to the identifier within the program using the function *definition*, we also retrieve the (extended) type of an identifier by using the function *typeof*, and we build a Damas-and-Milner-like assumption environment at the point the identifier is bound with the function *assumption-env*:

$$\begin{aligned} definition : &\ \ AExp \rightarrow Ide \rightarrow AExp \\ typeof : &\ \ AExp \rightarrow Ide \rightarrow TypeSch_{Ext} \\ assumption\text{-}env : &\ \ AExp \rightarrow Ide \rightarrow TEnv_{Ext} \end{aligned}$$

We use the function *ext* to build an initial "extended type" corresponding to a Hindley-Milner type as exposed in the section 4.1.1. Additionally, we use $inst_{ll}$, and $inst_{tv}$ to instantiate type variables, and outermost liveness labels; i.e., they act as universal quantifier eliminators. Their signatures are:

$$\begin{aligned} ext : &\ \ TypeSch \rightarrow TypeSch_{Ext} \\ inst_{ll}, inst_{tv} : &\ \ TypeSch_{Ext} \rightarrow TypeSch_{Ext} \end{aligned}$$

Finally, we use the 'closure' of an (extended) type $\hat{\tau}$ with respect to assumptions T, as

$$\overline{T}(\hat{\tau}) = \forall \theta_1 \ldots \theta_n . \hat{\tau}$$

where $\theta_1, \ldots, \theta_n$ are the liveness labels occurring free in $\hat{\tau}$, but not it T [DM82].

Functions identified with $\mathcal{T}$ implement the strategy for selection of nodes that was previously discussed, while those identified with $\mathcal{W}$ implement the actual extended type inference. The type signatures of all these are as follows[3]:

$$\begin{aligned} \mathcal{T}_p : &\ \ AExp \rightarrow AExp \\ \mathcal{T}^l : &\ \ FIde \rightarrow AExp \rightarrow TypeSch^l_{Ext} \\ \mathcal{T}^v : &\ \ VIde \rightarrow AExp \rightarrow TypeSch^v_{Ext} \\ \mathcal{T}_c : &\ \ 2^{VIde+FIde} \rightarrow AExp \rightarrow AExp \\ \mathcal{W}^l : &\ \ Funct \rightarrow Env \rightarrow TypeSch^l_{Ext} \\ \mathcal{W}^v : &\ \ Exp \rightarrow Env \rightarrow TypeSch^v_{Ext} \end{aligned}$$

[3]The operator *foldl* is a standard higher-order list operator of type

$$foldl :: (\beta \rightarrow \alpha \rightarrow \beta) \rightarrow \beta \rightarrow (List\ \alpha) \rightarrow \beta$$

defined as:

$$foldl\ f\ z\ [x_1, x_2, \ldots, x_n] = f\ (\ldots (f\ (f\ z\ x_1)\ x_2) \ldots)\ x_n$$

The reader is asked to read [BW88] for details

$$\mathcal{T}_p\ p\ =\ let\ p' = \alpha\text{-}rename\ p$$
$$G = dependency\ p$$
$$G' = collapse\text{-}cycles\ G$$
$$cs = topological\text{-}sort\ G'$$
$$in\ \ \mathcal{W}^v\ (foldl\ \mathcal{T}^e\ p\ cs)$$

$$\mathcal{T}^f\ f\ p\ =\ let\ (f(x_1^{\tau_1}, \ldots, x_n^{\tau_n}) = e) = definition\ f\ p$$
$$T = assumption\text{-}env\ f\ p$$
$$\hat{\tau}_1 = inst_{ll}\ (ext\ \tau_1)$$
$$\ldots$$
$$\hat{\tau}_n = inst_{ll}\ (ext\ \tau_n)$$
$$\hat{\tau} = \mathcal{W}^v\ e\ T[x_1 \mapsto \hat{\tau}_1, \ldots, x_n \mapsto \hat{\tau}_n]$$
$$in\ simplify\ (\overline{T}\ ((\hat{\tau}_1 \times \ldots \times \hat{\tau}_n) \rightarrow \hat{\tau}))$$

$$\mathcal{T}^v\ v\ p\ =\ let\ (v = e) = definition\ f\ p$$
$$T = assumption\text{-}env\ e\ p$$
$$in\ \ \hat{\tau} = \mathcal{W}^v\ e\ T$$

$$\mathcal{T}^e\ \{v_1, .., v_m, f_1, \ldots, f_n\}\ p = let\ \hat{\chi}_1^v = \mathcal{T}^v\ v_1\ p$$
$$\ldots$$
$$\hat{\chi}_n^f = \mathcal{T}^f\ f_n\ p$$

$$in\ \ if\ (\hat{\chi}_1^v = typeof\ v_1\ p)\ and\ \ldots\ and\ (\hat{\chi}_n^f = typeof\ f_n\ p)$$
$$then\ p$$
$$else\ \mathcal{T}_e\ \{v_1, .., v_m, f_1, \ldots, f_n\}\ p[v_1 \mapsto \hat{\chi}_1^v, \ldots, f_n \mapsto \hat{\chi}_n^f]$$

$$\mathcal{T}^e\ \{v_1, .., v_m, f_1, \ldots, f_n\}\ p = let\ \hat{\chi}_1^v = typeof\ v_1$$
$$\ldots$$
$$\hat{\chi}_n^f = typeof\ f_n$$
$$\ldots.$$
$$in\ \ldots$$

Function Designators

For function designators, we have the function $\mathcal{W}^f$ of the form

$$\mathcal{W}^f\ g\ T_i\ =\ case\ g\ of$$
$$clauses$$

where *clauses* are the following ones:

1. Functional Constants

$$
\begin{array}{l}
g = k_f^t \rightarrow \\
\quad let\ \hat{\chi}^f = Inst_{ll}\ Inst_{tv}\ (\mathcal{K}^f\ k_f) \\
\qquad S = \mathcal{PM}\ \hat{\chi}^f\ t \\
\quad in\ Subst\ \hat{\chi}^f\ S
\end{array}
$$

2. Functional Identifiers

$$
\begin{array}{l}
g = f^t \rightarrow \\
\quad let\ \hat{\chi}^f = Inst_{ll}\ Inst_{tv}\ (T_i\ f) \\
\qquad S = \mathcal{PM}\ \hat{\chi}^f\ t \\
\quad in\ Subst\ \hat{\chi}^f\ S
\end{array}
$$

Value Expressions

For value expressions, we also have the function $\mathcal{W}^v$ of a similar form as the above definition

$$
\begin{array}{lcl}
\mathcal{W}^v\ e\ T_i & = & case\ g\ of \\
 & & \quad clauses
\end{array}
$$

where *clauses* are the following ones:

1. Value Constants

$$
\begin{array}{l}
e = k_v^t \rightarrow \\
\quad let\ \hat{\chi}^v = Inst_{ll}\ Inst_{tv}\ (\mathcal{K}^v\ k_v) \\
\qquad S = \mathcal{PM}\ \hat{\chi}^v\ t \\
\quad in\ Subst\ \hat{\chi}^v\ S
\end{array}
$$

2. Value Identifiers

$$
\begin{array}{l}
e = v^t \rightarrow \\
\quad let\ \hat{\chi}^v = Inst_{ll}\ Inst_{tv}\ (T_i\ v) \\
\qquad S = \mathcal{PM}\ \hat{\chi}^v\ t \\
\quad in\ Subst\ \hat{\chi}^v\ S
\end{array}
$$

3. Applications

$$
\begin{array}{l}
e = g(e_1, \ldots, e_n) \rightarrow \\
\quad let\ (\forall \theta_{11} \ldots \theta_{1m_1} \ldots \theta_{n1} \ldots \theta_{nm_n}.\hat{\tau}_1 \times \ldots \times \hat{\tau}_n \rightarrow \hat{\chi}^v) = \mathcal{W}^f\ g\ T_i \\
\qquad \tau_1[\theta'_{11}/\theta_{11}, \ldots, \theta'_{1m_1}/\theta_{1m_1}] = \mathcal{W}^v\ e_1\ T_i \\
\qquad \ldots \\
\qquad \tau_n[\theta'_{n1}/\theta_{n1}, \ldots, \theta'_{nm_n}/\theta_{nm_n}] = \mathcal{W}^v\ e_n\ T_i \\
\quad in\ S\ \hat{\chi}^v[\theta'_{11}/\theta_{11}, \ldots, \theta'_{1m_1}/\theta_{1m_1}, \ldots, \theta'_{n1}/\theta_{n1}, \ldots, \theta'_{nm_n}/\theta_{nm_n}]
\end{array}
$$

4. Let Expression

$$e = \left(\begin{array}{l} let\ v_1^{\chi_1^v} = e_1; \\ \quad \ldots; \\ \quad v_m^{\chi_m^v} = e_m; \\ \quad f_1^{\chi_1^f}(x_{11}, \ldots, x_{1q_1}) = e'_1; \\ \quad \ldots; \\ \quad f_n^{\chi_n^f}(x_{n1}, \ldots, x_{nq_n}) = e'_n \\ in\ e \end{array} \right) \rightarrow$$

$$\mathcal{W}^v\ e\ T_i[v_1 \mapsto \chi_1^v, \ldots, v_m \mapsto \chi_m^v, f_1 \mapsto \chi_1^f, \ldots, f_n \mapsto \chi_n^f]$$

The extended type of all identifiers needed to compute the type of e are assumed to be already computed. This includes identifiers bound in the construct. This is consequence of performing the topological sort of nodes at function $\mathcal{T}_p$.

4.2.1 Example:

Consider the following program:

```
as = (Cons 1 (Cons 2 (Cons 3 Nil)))
append(xs,ys) = if (xs=Nil)
                  then ys
                  else Cons(head(xs),append(tail(xs),ys))
reverse(xs) = if (xs=Nil)
                then Nil
                else append(reverse(tail(xs)),Cons(head(xs),Nil))
in reverse(as)
```

The standard types of all let-bound identifiers of the preceeding program are:

$$\begin{array}{rc} \texttt{as}: & List\ Int \\ \texttt{append}: & \forall\alpha.(List\ \alpha) \times (List\ \alpha) \rightarrow (List\ \alpha) \\ \texttt{reverse}: & \forall\alpha.(List\ \alpha) \rightarrow (List\ \alpha) \end{array}$$

The initial extended type assignment for them is exactly as above, but interpreted as extended types. A topological order of the identifiers is {as, append, reverse}. Computing the extended type following this order leaves the type for as unchanged, the type for append converges after one iteration, and so does the type for reverse. The extended types are:

$$\begin{array}{rc} \texttt{as}: & List\ Int \\ \texttt{append}: & \forall\alpha.\forall\theta_1\theta_2\theta_3(List\ \alpha^{\theta_1}) \times (List^{\theta_2}\ \alpha^{\theta_3}) \rightarrow (List^{\theta_2}\ \alpha^{\theta_1,\theta_3}) \\ \texttt{reverse}: & \forall\alpha.\forall\theta(List\ \alpha^{\theta}) \rightarrow (List\ \alpha^{\theta}) \end{array}$$

That is, the result from append shares elements of both its argument lists, but only the second argument's structure is accessible from the result. For reverse, the result shares the elements of the argument list, but a new structure is created. Further, note that for the following version of reverse

```
reverse(xs) = if (xs=Nil)
              then xs
              else append(reverse(tail(xs)),Cons(head(xs),Nil))
in reverse(as)
```

the inferred type is

$$\texttt{reverse}: \ \forall\alpha.\forall\theta_1\theta_2(List^{\theta_1}\ \alpha^{\theta_2}) \rightarrow (List^{\theta_1}\ \alpha^{\theta_2})$$

i.e., even though the value of xs is known to be Nil at the then branch, this information is not known to the inferencer, which only knows about types, and not values.

4.3 Principal Typing

Lemma 1 All dependencies to a class have been solved before the collection is considered for extended-type inference $\mathcal{W}^f$, $\mathcal{W}^v$.

Proof This is a direct consequence of identifiers being classified into collections, and these collections being sorted topologically with respect to their interdependencies. □

Lemma 2 Suppose $\hat{\chi}_1^v = \mathcal{W}^v\ Te$, and $T \vdash e : \hat{\chi}_2^v$. Then $\hat{\chi}_2^v \succeq \hat{\chi}_1^v$. A similar lemma holds for the relation of $\mathcal{W}^f$ and $\overset{f}{\vdash}$.

Proof In general, $\mathcal{W}^v$ will assume a (precomputed) approximation to the type for every let-bound identifier defined inside e (rather than recursively computing their extended type). Even when the types assumed are the actual types, it can be proven by structural induction that the computed type can be instantiated to any type inferred by the inference rules. □

Lemma 3 Suppose $T \vdash e : \sigma$, σ being standard Hindley-Milner type. Then there is a finite lattice of extended types corresponding to σ, which only use free liveness labels appearing in T, and new bound liveness labels.

Proof By structural induction on the structure of the type of the expression. Type is finite, and hence it cannot contain an infinite number of new labels (there can be at most one new label per type node). □

Lemma 4 Let $\{v_1, \ldots, v_m, f_1, \ldots, f_n\}$ be a collection of let-bound identifiers of the expression e, and further assume that under assumptions T all (non-recursive) dependencies of this collection have been successfully typed. Further, assume that the types $\{\chi_1^v, \ldots \chi_m^v, \chi_1^f, \ldots \chi_n^f\}$ is the conjunctive fixpoint for this set of identifiers. Then these types can be inferred by appropiate application of the inference rules.

Proof This is proved by using induction on the structure of the expression. □

Theorem 1 $\mathcal{T}_p$ e computes the principal type for each let-bound identifier in the expression e.

Proof This is a consequence of the previous lemmas. □

5 Conclusion

For first order closed expressions, the extended type system is just a simple extension of the Hindley-Milner type system. Even though the types presented here are non-shallow on liveness labels, there are corresponding shallow versions for first order type. The reason is that the scope of all quantified variables extend to the right of the type (the intersection of the scopes of two variables is always one of them), therefore, a simple 'lifting' of the quantifiers to the outside of the type, with a suitable renaming of labels suffixes to transform them to a shallow type. On the other hand, extended types for the higher-order case cannot be made shallow, and it would be a mistake to try to do so.

On results related to type inference, we have shown that a 'most-general extended-type' exists for the first-order language presented here.

References

[Bak90] Henry G. Baker. Unify and conquer (garbage, updating, aliasing, ...) in functional languages. In *Proceedings of the 1990 ACM Conference on Lisp and Functional Programmming.* ACM, 1990.

[BW88] R. Bird and P. Wadler. *Introduction to Functional Programming.* Prentice Hall. 1988.

[DM82] L. Damas and R. Milner. Principle type schemes for functional languages. In *9th ACM Symposium on Principles of Programming Languages.* ACM, August 1982.

[GH90] Juan Carlos Guzmán and Paul Hudak. Single-threaded polymorphic lambda calculus. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*. IEEE, 1990.

[GL86] D.K. Gifford and J.M. Lucassen. Integrating functional and imperative programming. In *Proceedings of 13th ACM Symposium on Principles of Programming Languages*. ACM, 1986.

[Hin78] R. Hindley. The principal type scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1978.

[JG89] Pierre Jouvelot and David K. Gifford. Communication effects for message-based concurrency. Technical Report LCS TM-386, MIT, February 1989.

[KM89] Tsung-Min Kuo and Prateek Mishra. Strictness analysis: A new perspective based on type inference. Technical Report 89/13, The State University of New York at Stony Brook, 1989.

[LG88] J.M. Lucassen and D.K. Gifford. Polymorphic effect systems. In *Proceedings of 15th ACM Symposium on Principles of Programming Languages*. ACM, 1988.

[Mil78] R.A. Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348–375, 1978.

RESUMENES

Comité Modificado. Una Proposición de Red Neuronal para Clasificaciones y Decisiones

Prof. Wilmer González V
Escuela de Ingeniería Eléctrica
Universidad Central de Venezuela
Caracas. Venezuela

RESUMEN

Se propone una arquitectura de Red Neuronal de entradas continuas y salidas binarias, cuyos elementos tienen como función no-lineal la función signo. La proposición se presenta aquí para la clasificación de patrones correspondientes a contornos segmentados de imágenes binarias, sin embargo esta arquitectura puede utilizarse en otro tipo de reconocimiento y en la solución de problemas cuyo objetivo sea un conjunto de decisiones de acuerdo a la situación representada por el vector de entrada.

Al comportamiento global de la red contribuyen mecanismos de inhibición, que dan la posibilidad a algunas neuronas de manifestar un cierto estado de incertidumbre. La red propuesta representa un algoritmo sencillo y rápido de clasificación cuando es simulado en un computador digital-secuencial. Por otra parte, la red puede ser fácilmente construída para algunas aplicaciones en tiempo real, mediante un circuito mixto con componentes analógicos y digitales.

Integrando Técnicas de Sistemas de Gerência de Bancos de Dados e Sistemas baseados em conhencimiento para a construção de Sistemas de Informação Não-Convencionais

Mara Abel
Prof. do Instituto de Informática
Universidade Federal do Rio Grande do Sul
Av. Osvaldo Aranha, 99. CEP 90210
Porto Alegre. Brasil
Fax: (0512) 24-4164
e-mail: marabel@SBU.UFRGS.ANRS.BR

RESUMO

Sistemas de Gerência de Informações Não-convencionais representam uma classe de sistemas que manipulam volumes de informações distribuídas de diferentes tipos, como aquelas utilizadas por sistemas de apoio à gerência empresarial, de auxílio a projeto, sistemas cartográficos, médicos ou geológicos. O sistema prevê o tratamento de informações complexas, obtidas de fontes heterogêneas com comportamento dinâmico e, possivelmente, necessitando de procedimentos inteligentes de recuperação. O modelo aquí apresentado propõe a integração de tecnologías obtidas de Sistemas Baseados em Conhecimento e Sistemas de Gerência de Bancos de Dados de modo a obter o processamento eficiente de dados mais conhecimento, a través de sistemas modulares e independentes do conhecimento. São analisadas ainda as limitaçãpes do sistema proposto e as futuras direções da pesquisa na área.

Extensiones del MER y Lenguaje Gráfico para MER/PAC

Sergio Quezada González
Universidad de Santiago de Chile
Casillla 5659 Correo 2 - Santiago - Chile
FAX (562) 6811422

RESUMEN

En este trabajo un lenguaje gráfico para el Modelo Entidad Relacionamiento en Proyectos asistidos por computador (MER/PAC) es propuesto, como también una extensión al MER.

La definición del Lenguaje es realizada a través de un lenguaje de especificación de proyectos que posee estructuras de control semejantes a las ofrecidas en lenguajes de alto nivel.

El lenguaje permite la definición y manipulación de una base de datos en el contexto MER/PAC.

El sistema de diálogo que la interface ofrece como instrumento de comunicación al usuario, después de procedimientos de consistencia, se comunica con GERPAC que a través de SIGA accesa la base de datos.

A Database Perspective to SDL

Shem J. Ochuodho*
Department of Computer Science
University of York
e-mail: shem@minster.york.ac.uk

ABSTRACT

Although SDL[1] was originally intended for telecommunication systems, it has found use in other applications. However, that database community is one group which has seemingly not been endeared to SDL. It is not that database systems cannot be specified or described using SDL. Indeed the structural and behavioural concepts in SDL would be very useful for specifying database systems. We attribute the apparent lack of interest to the inherent "procedural" nature which SDL has been presented, and propose a database view. Introduced in this paper, is a system which adopts this perspective in addressing some problems inherent in communications software evolution. PVC-M uses an object-oriented variant of SDL, XDL.

*Usual address: Institute of Computer Science, University of Nairobi, P.O. Box 30197, Nairobi, Kenya

[1]CCITT's recommended specification and description language for telecommunication systems.

El Paradigma de los Orientados a Objetos usando un Enfoque Adecuado

Nidia Fernanda Arriazu
Rivadavia (oeste) 27
CP 4200 Santiago del Estero
República Argentina

RESUMEN

En este trabajo se enunciará informalmente un modelo de Ciclo de Vida para un producto Orientado a Objetos en el que simplemente se pretenden enunciar los pasos aconsejables a seguir para lograr dicho producto, posibilitando que el mismo goce de un cierto grado de calidad, fundamentalmente dado por el buen aprovechamiento de las ventajas de lo Orientado a Objetos.

El modelo que a continuación se enuncia no es más que el derivado de la experiencia adquirida al programar con lenguajes Orientados a Objetos. Dicho modelo del Proceso de Desarrollo de Software (PDS) es informal y por lo tanto su valor es solamente descriptivo. El motivo con el cual fundamento esta informalidad es el que en cierta forma inspira la redacción del presente trabajo: brindar a aquellas personas, que pretenden experimentar el paradigma de los Orientados a Objetos, una introducción que los lleve a familiarizarse con el mismo, recorriendo un camino adecuado, garantizando la calidad en sus productos, evitándoles confusiones que surgen por la búsqueda natural de paralelismos con paradigmas convencionales, que mas bien atrasan el aprendizaje del mismo por recorrer caminos erráticos o bien, introducen una idea deforme de lo que realmente es el paradigma de los Orientados a Objetos.

Manejador de Objetos para Aplicaciones de Diseño

José Abasolo P.
Claudia Jiménez G.
Fernando Aguirre
Elsa L. Rodriguez
Departamento de Ingeniería de Sistemas y Computación
Universidad de Los Andes
Cra. 1E No. 18A-10
Bogotá -Colombia
Tlf: +57 1 286 61 85
Fax: +57 1 284 18 90
e-mail (bitnet) : cjimenez@andescol

RESUMEN

Los avances tecnológicos han permitido el surgimiento de nuevas aplicaciones (CAD/CAM, Automatización de Oficinas, ...) que requieren manejo intenso de datos persistentes de estructura compleja. El paradigma de Orientación por Objetos (OO) en Bases de Datos parece el más adecuado para satisfacer esos nuevos requerimientos. Este artículo describe el modelo y la implantación de la primera versión de un **Sistema Manejador de Objetos**, orientado a soportar aplicaciones de diseño (CAD), que se está desarrollando en la Universidad de los Andes. El sistema sigue el paradigma OO, tiene una arquitectura de procesos del tipo Cliente-Servidor, utiliza un SMBD Relacional como manejador de memoria secundaria y va a ser probado con una aplicación de Apoyo al Diseño de Vivienda Masiva (INDUCON).

INCORPORACION DE MANEJO DE INCERTIDUMBRE EN UNA MAQUINA DE INFERENCIA EXPERIMENTAL USANDO PROGRAMACION ORIENTADA A OBJETOS

Ing. **Rodrigo J. Ramírez G.** Ing. **Víctor S. Theoktisto C.**, M.S.E.

Oficina Metrópolis
Universidad Metropolitana
Fax: (02) 241 9575 Dist. Universidad, Autopista Petare-Guarenas
P.O. BOX 76.819 CARACAS 1070

RESUMEN

La obtención de un juego de alternativas óptimo con respecto a ciertos criterios en problemas de análisis de decisiones bajo incertidumbre se puede plantear y resolver por medio un sistema basado en reglas. En una Máquina de Inferencia, no limitarse a un método único para resolver el manejo de incertidumbre permite analizar una mayor cantidad de problemas. En lugar de codificarse rígidamente los formalismos de incertidumbre comunmente usados, sobre la Máquina de Inferencia implícita en el lenguaje PROLOG se establece un paradigma de Programación Orientada a Objetos, en el cual se definen clases, objetos y mensajes necesarios para manejar, en este caso, premisas, reglas e hipótesis. La representación del manejo de incertidumbre deseado se puede describir como el paso de mensajes entre dichos objetos a través de uno o más métodos. Esto permite incorporar sin mayores modificaciones a modelos ya hechos nuevos métodos de manejo de incertidumbre que responderán al mismo mensaje de evaluación de incertidumbre, suministrando de forma sencilla y a nivel académico una versatilidad extrema, disponible hasta ahora en sistemas muchísimo más caros. Como resultado gratuito de este desarrollo queda un manejador de clases, objetos y mensajes que puede ser usado para modelar otras situaciones.

Bases Estadísticas en las Bases de Datos

Marta Sananes
IEAC/FACES/Universidad de Los Andes
Núcleo La Hechicera, Ed. B, Mérida
Telf. (074) 401116. Fax: (074) 401014

RESUMEN

Muchas Organizaciones no cuentan con estadísticas apropiadas por falta de planificación, insuficiencia en sus Bases de Datos o por dificultades para reprocesar la información. Las Organizaciones que mantienen estadísticas no cuentan con las herramientas apropiadas; utilizan paquetes estadísticos (que no ayudan en la recopilación) y/o desarrollan aplicaciones para satisfacer los distintos requerimientos con lo cual tienden a presentarse en las estadísticas los mismos problemas de redundancia e inconsistencia del ambiente ordinario de archivos por aplicaciones.

Se propone un cambio de enfoque; bajar la recopilación estadística del alto nivel externo al bajo nivel interno de los DBMS, según especificaciones del DDL, los cálculos y presentaciones finales. El aprovechamiento de este enfoque supone en las Organizaciones el reconocimiento de la importancia de planificar sus estadísticas y analizar los requerimientos estadísticos a todos los niveles de la Organización, incorporándolos a los diseños de sus esquemas conceptuales.

Se propone además concretamente una extensión al SQL incorporando los criterios mencionados. También se reseñan brevemente algunos trabajos en los que se ha ensayado la viabilidad y utilidad de algunos de tales criterios.

Las Bases de Datos y el Algebra Relacional como Herramientas en la Construcción de Algoritmos que Analizan la Sintáctica y la Semántica de un Programa Escrito en seudocódigo

Francisco Ríos Acosta*
Javier Medina Zuniga†
Ana M. Carrillo C.‡
Sandra Martínez T.§
Ma. Gpe. Alvarado M.¶

RESUMEN

Se presenta un algoritmo recursivo que utiliza operaciones fundamentales del Algebra Relacional, para realizar el análisis sintáctico y semántico de un programa construído en seudocódigo. Se considera en la primera fase, que el programa escrito en seudocódigo, utiliza instrucciones de iteración, selección, asignación, entrada y salida. Se utilizan diagramas Entidad-Relación (de CHEN), para modelar el mundo de objetos existentes en un programa codificado en seudocódigo y las relaciones que los asocian. Las entidades y relaciones son obtenidas identificando las dependencias funcionales que existen entre los objetos del programa. Podemos decir que los algoritmos propuestos pueden ser trasladados a la programación orientada a objetos, debido al enfoque de base de datos basado en objetos.

*Universidad Autónoma Metropolitana. Unidad Atzcapotzalco
†Intituto Tecnológico Tlalnepantla
‡Intituto Tecnológico Tlalnepantla
§Intituto Tecnológico Tlalnepantla
¶Intituto Tecnológico Tlalnepantla

Ambiente JOYCE+ de desarrollo de Sistemas Distribuidos para entorno Unix TCP/IP

Alvaro Arenas
María Consuelo Franky*
Dpto. de Ingeniería de Sistemas y Computación
Universidad de Los Andes
Apartado Aéreo 4976
Fax +57 1 284 18 90
Bogotá - Colombia

RESUMEN

En este artículo se describen los componentes fundamentales de un Ambiente que apoya el desarrollo de sistemas distribuidos sobre redes de computadores UNIX comunicados bajo protocolo TCP/IP. El Ambiente está basado en el modelo y lenguaje JOYCE+ [Franky 90] el cual contempla la comunicación asincrónica y no confiable entre procesos que se ejecutan en diversos sitios. Esta comunicación entre procesos se realiza de manera indirecta a través de objetos "canales" permitiendo independizar la especificación de los procesos respecto al ambiente específico de implantación. El modelo JOYCE+ contempla además la activación dinámica de procesos haciéndolo apropiado para expresar problemas de procesamiento paralelo. El ambiente JOYCE+ que aquí se describe consta básicamente de un traductor que transforma la especificación JOYCE+ de un sistema distribuido a procesos expresados en lenguaje C y de un subsistema de comunicación que implanta los objetos canales los cuales asegurarán el intercambio de mensajes de los procesos en el entorno específico de computadores UNIX conectados en red bajo protocolo TCP/IP. En el futuro se planea utilizar el Ambiente JOYCE+ para construir sistemas distribuidos que faciliten el desarrollo de aplicaciones sobre Bases de Datos Distribuidas o sobre Archivos Distribuidos.

*e-mail: cfranfy@andescol.bitnet

An Educational Linear Programming Software Environment

Antonio Fernando Branco Costa
Edson Luiz França Senne
Fernando Augusto Silva Marins
São Paulo State University (UNESP)
Engineering School (FEG) - CP 205
12500 - Guaratinguetá - SP - Brazil

ABSTRACT

This paper describes a software environment that allows to test user's knowledge on Linear Programming traditional techniques. In this environment all interactions with the users are performed by means of an appropiated graphical interface. During a problem solving session, the computer guides the users through algorithms steps applications - evaluating the necessary computations - letting the users the decision-making process and warning them when a wrong decision is taken. The requirements for a software environment to support learning and teaching tasks are discussed. A module composing the environment is shown and an example of its use is given.

La Evaluación del Desempeño en una Metodología de Desarrollo de Sistemas de Información Distribuidos

José Aguilar*
Alberto Valderruten†

RESUMEN

La constante evolución de los sistemas de información distribuidos, aquellos que además de la noción clásica de sistema incluyen las características de repartición y comunicación, ha hecho que cada día sus arquitecturas sean mas complejas, y las restricciones a nivel de desempeño (tiempos de respuesta, plazos de transmisiones de mensajes, ...) jueguen un papel importante.

Los costos de construcción y mantenimiento de tales sistemas, hacen necesario el uso de herramientas cada vez mas sofisticadas. La utilización de herramientas que representen el sistema en estudio permite adquirir un mayor conocimiento de éste y validar su comportamiento a priori para asegurar el producto final.

En este artículo se presenta el interés de la evaluación de desempeño en el desarrollo de un sistema de información distribuido. Se describen los métodos de evaluación de desempeño que utilizan una representación del sistema para hacer su estudio, se propone una metodología de desarrollo de un sistema de información distribuido, se presentan los criterios de desempeño característicos de estos sistemas y el uso de la evaluación de desempeño en cada una de las etapas de la metodología propuesta. Finalmente, se especifica una herramienta de evaluación de desempeño para el desarrollo de una clase de sistemas de información distribuidos, los sistemas de comunicación.

*Grupo MaMuD. Departamento de Computación. Facultad de Ingeniería. Universidad de Los Andes. Mérida. Venezuela. *e-mail: aguilar@irit.fr*

†Institut de Recherche en Informatique de Toulouse. Equipe SIERA. Université Paul Sabatier. 118, Route de Narbonne 31062. Toulouse Cédex. France. *e-mail: valderru@irit.fr*

Scheduling of Tanker Fleet

M. Lentini*
A. Reinoza†
J.E. Smith‡

ABSTRACT

We describe a computational model that produces the monthly scheduling of a tanker fleet which transports oil products between a loading port and several unloading ports.

The model is represented by a sequential decision tree where a particular schedule is given by a path in the tree. We used an intelligent search algorithm to find an "optimal path" in this tree.

The software was developed for MARAVEN S. A., subsidiary of PDVSA (Venezuelan Petroleum Holding) and it is currently used to operate the fleet.

*Departamento de Matemáticas Puras y Aplicadas, Universidad Simón Bolívar. Apartado 89000. Caracas, Venezuela. FAX +58 2 962 16 95. *e-mail: sun!emsca!usb!mlentini*

†Departamento de Computación y Tecnología de la Información, Universidad Simón Bolívar & IMG Consultores S.A.

‡IMG Consultores S.A.

ASPYR-1: Un Arreglo Sistólico Programable y Reconfigurable

Germinal Isern
Giuseppe Sena
Rosy Leo
Universidad Central de Venezuela
Facultad de Ciencias,
Escuela de Computación
Apartado 47002. Caracas, Venezuela.

RESUMEN

En este trabajo se presenta el diseño de un computador sistólico que puede ser utilizado para una amplia gama de aplicaciones. La capacidad de ser programable es la característica principal de esta máquina, de tal forma que la operación a ser realizada por cada EP[1] pueda variar dinámicamente de acuerdo a la aplicación a ser desarrollada. Además, la red de EP es reconfigurable; permitiendo así modificar los enlaces físicos de comunicación entre los EP para adoptar la topología más idónea al tipo de algoritmo a ser ejecutado. Otro aspecto importante es la modularidad y expandibilidad del arreglo, de manera que sea posible incrementar la capacidad de procesamiento de la máquina para que se ajuste eficientemente al tamaño del problema. El diseño físico de esta arquitectura se realizó en torno a los componentes de la familia transputer.

[1]Elemento de Procesamiento

Introducción a la Paralelización de Algoritmos para Problemas de Ubicación-Distribución en el Plano

Alejandro Crema
Escuela de Computación. Facultad de Ciencias U.C.V.
Apartado: 47002. Caracas, Venezuela
Fax: +58 2 662 71 21

RESUMEN

El propósito de este trabajo es presentar una alternativa de Algoritmos en Paralelo para problemas de Ubicación-Distribución en el Plano. Se identifican dos sub-problemas auxiliares: el primero de Programación Lineal ó Lineal-Entera; el segundo de Programación No-Lineal consistente de un conjunto de problemas independientes. La idea fundamental del Algoritmo secuencial es alternar métodos que resuelven los dos subproblemas hasta lograr soluciones que cumplen condiciones necesarias de optimalidad. El paralelismo surge al explotar la independencia de los problemas No-Lineales y en la resolución del problema de Programación Lineal ó Lineal Entera.

A los efectos de implantación el estudio se restringe al problema clásico de Asignación y Distribución en el Plano. Se presentan resultados computacionales comparativos entre la visión secuencial y una implantación en paralelo sobre un simulador de la arquitectura Hipercubo.

Paradigmas de Desenvolvimento de Interfaces do Usuario

Afonso Inácio Orth*
Daltro José Nunes†

RESUMO

O presente artigo tem como objetivo fazer uma análise exaustiva da comunicação homem/máquina (interface do usuário), apresentar uma taxonomia para interfaces e, finalmente apresentar de maneira informal um modelo de estruturação de software, implementável por meio do paradigma da orientação a objetos, com destaque para interface do usuário.

*COPPE/UFRJ. Mestre em ciência de Computação pela UFRGS e doutorando em Engenharia de Software na COPPE/UFRJ

†Doutor em Informática pela Universidade de Stuttgart. Rua Olavo Bilac, 126/503. 90.040 - Porto Alegre - RS - Brasil

¿La Programación Declarativa admite una puesta a punto Declarativa?

Lic. Javier Díaz
Lic. Patricia Bazán
Laboratorio de Investigación y Formación en Informática Avanzada
Departamento de Informática Facultad de Ciencias Exactas
Universidad Nacional de La Plata
Buenos Aires, Argentina

RESUMEN

El desarrollo de software como actividad humana orientada a la resolución de problemas tiene muchos aspectos de interés. En este artículo se describen las alternativas para corregir o ajustar los resultados de un programa a lo esperado.

De esta forma el objetivo que se persigue, la puesta a punto de programas lógicos, la metodología está basada en heurísticas (que varían según el programa, el dominio de aplicación) que tornan difícil de sistematizarse.

La puesta a punto de programas lógicos admite múltiples visiones y encierra dificultades para mantener un adecuado nivel de abstracción (por parte del diseñador) y simultáneamente interpretar posibles causas de comportamiento erróneo (inesperado).

Diseño y Realización de un Interpretador Morfo-Sintáctico Universal para Ciertos Lenguajes Naturales y Artificiales

Luc B. du Boucheron
Clara Baldi Costa
Nelson Fernández
Robert Hutchinson
Universidad Central de Venezuela
Facultad de Ingeniería, Caracas

RESUMEN

Un grupo de profesores de la Facultad de Ingeniería de la UCV, interesados en la producción de software de apoyo para la docencia, está desarrollando un analizador morfo-sintáctico que permite reconocer si una oración simple o una expresión está escrita correctamente o no. El programa no está destinado a un solo lenguaje; hasta ahora, han sido incorporados el inglés, el español y el álgebra.

Para cada lenguaje, se elaboran varios diccionarios y dos grupos de reglas: las reglas morfológicas y las reglas sintácticas. Las reglas están escritas según un modelo que permite la transferencia de la información necesaria al análisis morfológico y sintáctico así como datos de carácter semántico.

Se están diseñando actualmente varios proyectos que podrían hacer uso de este analizador. Uno de ellos está relacionado con la enseñanza del inglés con computadoras. En otro proyecto, se pretende producir un programa interactivo que puede simular la geometría euclideana plana.

El análisis sintáctico de una oración produce una estructura que puede servir de base para la obtención de información semántica. Esto debe permitir el uso de este programa para que un robot interprete y ejecute una orden emitida en un lenguaje casi natural.

Expert System for Priority Treatment of Public Transport at Signalised Intersections

Carlos David Nassi
Programa de Engenharia de Transportes
COPPE/UFRJ
Caixa Postal 68512
CEP 21945 Rio de Janeiro. Brasil
Tlf: (55) (21) 270-3697
Fax: (55) (21) 290-6626

ABSTRACT

This paper explains the expert system used to help professionals in the Public Transport and Traffic Engineering fields to implement priority treatment for public transport at signalised intersections.

Most of the techniques available to give priority treatment for public transport have the minimisation of travel time as its main purpose.

Besides those common techniques which treat road infraestructure, there are a number of others which are used at signalised intersections. Those techniques are: cycle time or phase changes (for fixed cycle time), public transport priority by selective detection (for non-fixed cycle time), and exemption for public transport from bans on turning. However, criteria for decision making to choose the best technique is not very well treated.

Those three techniques have some different details. For example, the use of extension, recall, hurry-call or combinations for those possibilities are also incorporated.

The expert system helps professionals to choose the best treatment for each different intersection in order to achieve two possible purposes: public transport passenger delay minimisation, or public transport passenger delay minimisation without increasing of delay for private transport users.

Those techniques could be useful whether developed or developing countries due to congestion problems and, for developing countries due to increasing demand for public transport.

Aplicação das Teorias de Dempster-Shafer e Computação Local na Avaliação da Tecnologia do Biogas

Aurora A. Kawahara*
Peter M. Williams†

ABSTRACT

Neste trabalho aplicamos as teorias de Dempster-shafer e de computação Local na avaliação da tecnologia do biogas nas areas rurais do Brasil. Discutimos dois estudos de caso em detalhe e comparamos os resultados com um método de avaliação de projeto mais tradicional. Do lado computacional mostramos como a computação local e a redução da dimensão em casos onde relações entre variaveis são válidas sem exceção, podem aumentar a eficiencia.

*Coppe/UFRJ
†University of Sussex

Un Sistema Basado en Conocimiento para la Planificación de Tripulaciones*

Andres Vinet
Eduardo Parra
Miguel Nussbaum
Departamento de Ciencia de la Computación
Escuela de Ingeniería
Pontificia Universidad Católica de Chile
Casilla 6177, Santiago - Chile
FAX 56 2 552 4054
mn@puc.cl

RESUMEN

Un problema cotidiano en las aerolíneas es la asignación de tripulaciones a cada uno de los vuelos existentes. La asignación debe ser tal, que respete todas las normas existentes como son entre otros los tiempos de vuelo, los días de descanso, etc. La cantidad de tripulantes involucrados, así como la de vuelos y las restricciones existentes, hace muy tedioso y demoroso el proceso de planificación, fuera de ser en la mayoría de los casos imposible el obtener una solución que siquiera se acerque al óptimo. Este trabajo presenta una solución a este tipo de problemas que utiliza las técnicas de los sistemas basados en conocimiento, la cual fue implementada para una línea aérea de mediano tamaño.

*Este trabajo fue parcialmente financiado por el Fondo Nacional de Ciencias (FONDECYT) proyecto 90-0765.

Uma Arquitetura com Plano de Controle para Sistemas Especialistas

Edson Luiz França Senne*
Célia Barbosa Ghedini†

RESUMO

Este trabalho propõe uma nova arquitetura para sistemas especialistas desenvolvidos a partir de uma linguagem de engenharia de conhecimento baseada em regras de produção. A arquitetura proposta inclui um plano de controle para estruturar conceitualmente a base de conhecimento de um sistema especialista, facilitando o seu desenvolvimento e manutenção. O plano de controle libera o especialista de preocupações com o controle de regras e permite o direcionamento de blocos de regras específicos ao mecanismo de inferência, mantendo todas as características desejáveis a um sistema baseado em conhecimentos: modularidade, flexibilidade, extensibilidade e capacidade adaptativa. Como resultado da utilização do plano de controle são possíveis explanações coerentes do raciocínio do sistema durante o processo de resolução de problemas.

*Universidad Estadual Paulista (UNESP). Faculdade de Engenharia. Caixa Postal 205. 12500. Garatinguetá. SP. Brasil

†Instituto Tecnológico de Aeronáutica. Departamento de Ciência da Computação. 12200. São José dos Campos. SP. Brasil

Hacia una Lógica Subjetiva

Eric Fimbel
Departamento de Computación y Tecnología de La Información
Universidad Simón Bolívar

RESUMEN

Deseamos utilizar la información CUALITATIVA y SUBJETIVA que expresan los seres humanos ("cilindro muy caliente", "ojo del paciente amarillento", etc.). Para ello, estamos desarrollando un sistema formal de tipo Lógica de primer orden que permite expresar apreciaciones, predicados modulados ("muy caliente") y comparaciones ("cilindro #1 más caliente que cilindro #2").

Cada expresión se traduce en un conjunto de ecuaciones simbólicas (Ej. CALOR(C) $\approx$ MUY(CALIENTE)). Utilizamos exclusivamente los operadores "O", "NO", "=", "$\approx$", "$<$" y los símbolos "0", "$+\infty$", "$-\infty$". Jamás aparecen números en las ecuaciones.

Mediante Axiomas particulares del lenguaje, Axiomas generales y reglas de producciones, derivamos nuevas ecuaciones que retraducimos en apreciaciones y comparaciones. El sistema tiene dos propiedades importantes:

1) Infiere lo mismo que un ser humano. Ej.: "cilindro #1 caliente" y "cilindro #2 muy caliente" implican que "cilindro #2 más caliente que cilindro #1". En cambio, "paciente muy caliente" no implica que "paciente más caliente que cilindro #1".

2) Procesa información subjetiva. Solamente utiliza el hecho que "calientísimo" es más fuerte que "caliente"; no necesita conocer la fuerza exacta que el locutor le da a ambas palabras.

El prototipo corre sobre la PC. Utiliza un algoritmo parecido a la resolución para contestar preguntas.

Critérios de Avaliação e Modalidades do uso do Computador como Tutor

Gilda Helena Bernardino de Campos*
Ana Regina C. da Rocha†

RESUMO

A diversidade dos usos que o computador pode ter em educação tem suscitado questões relativas a avaliação do software educacional.

Dois importantes focos de estudo tem sido objeto de pesquisa de especialistas na área de informática em educação: (a) a taxonomia da utilização educacional dos computadores e, (b) a avaliação do software educacional.

Este trabalho identifica, considerando o domínio do uso do computador como tutor, quais os critérios mais relevantes a serem considerados na avaliação do producto em suas diversas modalidades. Relaciona estes critérios com modelos teóricos do "design" instrucional que podem ser utilizados neste tipo de aplicação do computador em educação.

*CIES/EDUCOM e COPPE/UFRJ. Rua Schmidt Vasconcellos 83/501. Cosme Velho. 22241. Rio de Janeiro. R.J. Brasil
†COPPE/UFRJ. Caixa Postal 68511. 21945. Rio de Janeiro. R.J. Brasil

Una Experiencia en la Enseñanza de la Computación aplicada al Diseño y el Dibujo (C.A.D.D.) para los Estudiantes de Arquitectura en la Universidad Simón Bolívar

Sowiesky Galavís

RESUMEN

Este trabajo presenta el desarrollo y aplicación de una experiencia docente mediante la cual los estudiantes de arquitectura (y próximamente los estudiantes de urbanismo) en la Universidad Simón Bolívar ven incorporado a su curriculum académico la computación aplicada al diseño y al dibujo (C.A.D.D.)

Se especifica el por qué del Software seleccionado y se analizan los resultados en términos de logros obtenidos (presentandose el avance en el uso de las herramientas de diseño y dibujo por horas de clase recibidas) y niveles de satisfacción por parte de los estudiantes.

Se concluye que esta es una experiencia altamente positiva que seguramente abrirá fronteras a nuestros egresados en el ejercicio de su profesión.

SEDILAG 1.0^{tm}
Sistema Experto para el diagnóstico del comportamiento de los pozos que producen bajo el esquema de levantamiento artificial por gas

Edicson León
Dáquer Reverón
Alberto Prato

RESUMEN

SEDILAG 1.0^{tm} es un sistema experto que tiene como objetivo apoyar al personal de Ingeniería de Producción en su labor de diagnóstico del comportamiento de la producción de los pozos que producen bajo el método de Levantamiento Artificial por Gas *(LAG)*. Este sistema proporciona asesoría experta sobre posibles fallas en el equipo de levantamiento artificial por gas, del pozo, y causan mermas en su producción de crudo.

En la actualidad este esquema de producción es de vital importancia dentro de la División de Occidente, donde el 80% de la producción de crudo se extrae mediante la inyección de gas con un número de pozos asociados que supera los 4000.

Con la incorporación de *SEDILAG* 1.0^{tm} se optimiza la tarea de diagnóstico, mejorando la calidad y reduciendo el tiempo requerido para su realización. Todo esto dentro de un marco de mejoramiento constante en la calidad del trabajo y preparación para afrontar los retos de incremento de producción que nos depara el siglo XXI.