4.0
17
91
I

CEDINFOR

PANEL'91

XVII Conferencia Latinoamericana de Informática

CNIASE'91

IV Conferencia Venezolana de Inteligencia Artificial y Sistemas Expertos

Volumen I

8 al 12 de Julio de 1991
Caracas
Venezuela

FUNDACION CEDINFOR
ORIGEN DOCUMENTO SALDO
NRO. INV. 1242
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

SIEN: 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

PARALELIZACION DE UN PROBLEMA DE APLICACION SOBRE UNA PLATAFORMA DE TRANSPUTADORES

Héctor Klíe[†] y Alejandro Teruel[‡]

Resumen. Se estudia la paralelización de un programa de frecuente uso en geofísica llamado DMO ("Dip-moveout"). El DMO posee tres niveles principales de paralelismo: por secciones sísmicas, por trazas sísmicas, por integración de los datos. Se implantan los dos últimos niveles en una malla de procesadores tipo transputadores ("transputers") aplicándose técnicas de descomposición de dominios para lograr un alto grado de paralelismo y analizándose sus comportamientos tanto teóricos como experimentales. El caso de la integración de los datos se reformula como un producto matriz-vector, en donde se presentan una serie de alternativas derivadas de los diferentes esquemas secuenciales: Fila, columna y bloque. El primero de estos aportó los mejores resultados desde el punto de vista tanto teórico como experimental.

1. Introducción

El "dip-moveout" (DMO) es una aplicación geofísica que involucra grandes cantidades de datos y adicionalmente, consume mucho tiempo de procesamiento. Entre los métodos para efectuar el DMO se destaca el de Hale [10][11], el cual se puede enunciar a través de las siguientes ecuaciones:

$$p_0(\omega_0, k, h) = \int \mathrm{A}^{-1} e^{i\omega_0 t \mathrm{A}} dt \int p(t, x, h) e^{-ikx} dx \tag{1}$$

$$p_0(t, x, h) = \frac{1}{4\pi^2} \int e^{-i\omega_0 t} d\omega_0 \int p_0(\omega_0, k, h) e^{ikx} dk \tag{2}$$

donde $A = \left(1 + \frac{k^2 h^2}{\omega_0^2 t^2}\right)^{\frac{1}{2}}$; $p(t, x, h)$ representa el sismograma original; $p_0(t, x, h)$, el sismograma corregido; t, el tiempo; x, la distancia y h el "offset".

La ecuación (1) indica una transformada de Fourier sobre x, y luego una integración sobre t para todo ω_0 y k. La ecuación (2) se traduce en una doble transformada de Fourier inversa sobre k y ω_0. Para realizar las transformadas de Fourier se utiliza el conocido algoritmo de la transformada rápida de Fourier (FFT), cuyo orden es $n \ln n$ para un conjunto de n datos.

[†] Departamento de Informática, Sección de Tecnología, INTEVEP S.A.

[‡] Departamento de Computación y Tecnología de la Información, Universidad Simón Bolívar

El cálculo del DMO es computacionalmente muy costoso, debido a la integral sobre t, cuyo orden es n^2. Se han realizado intentos para eliminar la componente computacional de dicha integral [11]. Sin embargo, los esfuerzos al respecto han conducido a simplificaciones que perjudican notablemente la precisión del método. En la Fig. 1 se muestra en detalle el fragmento correspondiente a la integración en el DMO. Nótese que la complejidad del fragmento, y por ende del algoritmo, está dado por $O(H*N*M^2)$.

```
/* N es el numero de trazas sismicas */
/* M es el numero de muestras */
/* H es el numero de secciones sismicas */
Para cada "offset" h perteneciente a {h1,..hH}
....
aux:= SQR(M / (dx * N))
/* Ciclo sobre el numero de onda (trazas transformadas) */
FOR ik:=0 TO N-1 DO
  IF ik > (N / 2) THEN k:= N - ik
                  ELSE k:= ik
/* Ciclo sobre las frecuencias (muestras transformadas) */
  FOR w:=0 TO M-1 DO
    IF w > (M / 2) THEN w0:= M - w
                   ELSE w0:= w
    Transf:=(0.0, 0.0)
/* Ciclo sobre el tiempo (muestras sin transformar)*/
    FOR t:=0 TO M-1 DO
/* Calculo del operador DMO */
      A:=SQRT(1+aux*SQR(k*h/(t*w0))
      fase:= 2*Pi*w0*t*A
 /* Calculo de la exponencial compleja */
      Parte.real:= Cos(fase)
      Parte.ima:= Sen(fase)
 /* Conversion a complejos */
      Complejo:= Complex (Parte.real, Parte.ima)
      Transf:= Transf + Complejo*P(t,ik,h) / A
    P0(w,ik,h ):= Transf
  ....
```

Fig. 1. Fragmento en pseudo-Pascal que describe el algoritmo secuencial de la integral en el DMO para corregir H secciones sísmicas.

El presente trabajo se centra en estudiar la paralelización del algoritmo de DMO propuesto por D. Hale. A lo largo del trabajo, se podrá observar la riqueza de puntos de tratamiento que posee el algoritmo. De hecho, se explorarán diferentes niveles de paralelismo pero haciendo énfasis en la descomposición de dominios que se pone de manifiesto en el costoso cálculo de la integral ya mencionada.

En la sección 2, se describen brevemente la arquitectura y las métricas utilizadas. En la sección 3, se discuten los niveles de paralelismo detectados en el algoritmo de Hale. En las secciones 4 y 5 se estudiarán los dos últimos niveles de paralelismo encontrados. En la sección 6, se presentan y discuten los resultados experimentales sobre la base de transputadores.

Finalmente, para medir los alcances de las ideas propuestas, se ofrecen una serie de conclusiones y direcciones futuras de trabajo en la sección 7.

2. Reseña de la arquitectura y de las métricas adoptadas

2.1 Topología de malla

Para el estudio de los algoritmos y las pruebas experimentales se partirá del hecho de que se cuenta con una malla de $P1 \times P2$ transputadores. Adicionalmente, el transputador $P_{1,1}$ funge como el único punto de entrada y salida de los datos.

Tanto $P1$, $P2$ y el número de las muestras M se consideraran potencias de 2, con lo cual se asegura la divisibilidad de M entre P=$P1$x$P2$ para $M \geq P$. Durante el curso de este trabajo se tuvo acceso a una plataforma de 8 transputadores con 1 Mbyte de memoria cada uno y 1,5 Mflops de desempeño pico.

2.2 Cómputo y comunicación

Se utiliza la siguiente notación:

- $T_{cal}(M, P)$, es el tiempo de cómputo sobre P procesadores.
- $T_{com}(M, P)$, es el tiempo de comunicación de datos entre los procesadores.
- $T_o(M, P)$, como el "overhead" de comunicación y sincronización.

El tiempo de envío de un dato compuesto de m bytes entre dos procesadores vecinos está dado por :

$$t_{start} + mt_{send} \tag{3}$$

donde t_{send}, es el tiempo de envío de un byte y t_{start}, es el tiempo de latencia antes de enviar efectivamente el mensaje. El lector puede observar que en la comunicación entre nodos vecinos $T_o(M, P) = t_{start}$ y $T_{com}(M, P) = mt_{send}$.

2.3 Aceleramiento

El aceleramiento ("speedup") es una función del número de procesadores P y del tamaño del problema M [1] que está expresado por :

$$A(n, p) = \frac{T(n, 1)}{T(n, p)} \tag{4}$$

2.4 Fracción serial

A. Karp y H. Flatt [14] exponen la siguiente métrica:

$$f = \frac{\frac{1}{A(n,p)} - \frac{1}{p}}{1 - \frac{1}{p}} \tag{5}$$

3. Niveles de paralelismo asociados al problema

En la Fig. 2 se ilustra la relación entre los elementos que conforman un sismograma de sirve de entrada al procesamiento DMO.

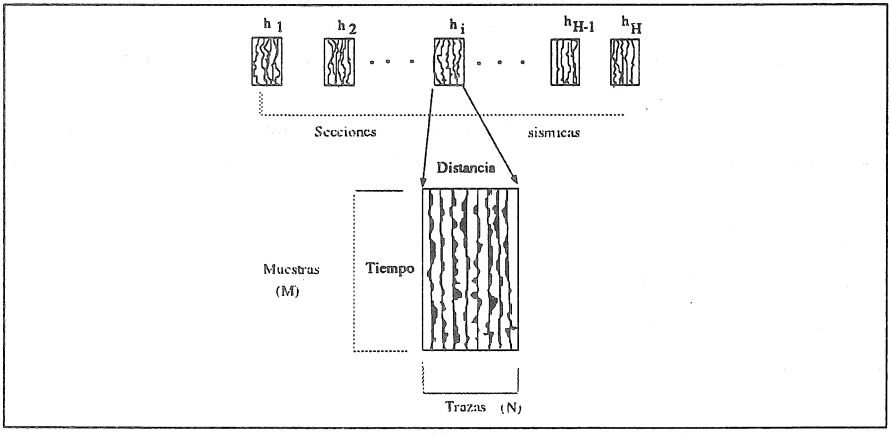


Fig. 2. La sección sísmica vista como un arreglo de datos bidimensional de tamaño MxN.

En primer término el sismograma se compone de conjuntos diferenciados de trazas que reciben el nombre de secciones sísmicas. Como en las ecuaciones (1) y (2) el parámetro h permanece constante, cada sección sísmica con un h dado puede ser procesada independientemente de las demás.

Al propio tiempo, cada sección es un arreglo bidimensional de datos que es transformado en forma independiente a lo largo de sus parámetros o ejes (tiempo y distancia), tal como se desprende de las integrales en (1) y (2). De modo que para cada traza (muestra) se pueden transformar simultáneamente todas las muestras (trazas) de una sección sísmica en particular. De esta manera, una misma sección se puede dividir en lotes de trazas o en lotes de muestras (i.e. cortes verticales u horizontales) y efectuar la correspondiente transformada de Fourier o integración de un grupo determinado de datos.

Otra descomposición del dominio del problema tiene que ver con la costosa integración sobre t en (1). Para ello, el fragmento de la integración se expresa matricialmente de la siguiente manera:

$$\begin{pmatrix} P_0(0,k,h) \\ \cdot \\ \cdot \\ P_0(M-1,k,h) \end{pmatrix} = \tag{6}$$

$$= \begin{pmatrix} W^{0,0} & \cdot & \cdot & W^{0,M-1} \\ \cdot & W^{1,1} & \cdot\cdot & \cdot \\ \cdot & \cdot & \cdot\cdot & \cdot \\ W^{M-1,0} & \cdot & \cdot & W^{M-1,M-1} \end{pmatrix} \begin{pmatrix} P(1,k,h) \\ \cdot \\ \cdot \\ P(M-1,k,h) \end{pmatrix}$$

donde $W^{wt} \equiv e^{2\pi i w_0 t \Lambda(w_0,t,k,h)} \mathrm{A}^{-1}(w_0,t,h,k)$.

La reformulación anterior ilustra como el nivel de paralelismo por integración de los datos aprovecha la descomposición del producto matriz-vector entre los procesadores.

La discusión anterior, basada sobre la Fig. 2 apunta a los diferentes niveles de paralelismo encuentrados en el problema del DMO:

- Secciones sísmicas
- Trazas sísmicas
- Integración de los datos.

El primer nivel no se pudo implantar en la plataforma disponible por falta de memoria [15]. En las secciones siguientes se analizan las ventajas y desventajas computacionales que trae consigo la exploración de los dos últimos niveles.

4. Paralelismo por trazas sísmicas

Una sección sísmica puede subdividirse en lotes de trazas. Esto permite

- Paralelización sencilla del programa .- Solamente se requiere la paralelización del lazo más externo de la integración (i.e. el lazo sobre *ik* de la Fig. 1). Por tanto, basta tomar los datos pasados por la primera transformada de Fourier y devolver los que resultan de la integración.
- Menor requerimiento de memoria para la implantación.- Bastaría utilizar dos arreglos de complejos de dimensión $\frac{NM}{P}$ para almacenar los datos antes y después de la integración. Para datos típicos (i.e. N=128, M=2048), esto implica un requerimiento de al menos 5 transputadores para correr la aplicación.
- Manejo fácil de la entrada y salida.- Basta hacer una carga y posterior escritura de la sección sísmica a corregir, tal como se hace en la versión secuencial. De esta manera se evita el manejo concurrente de grandes volúmenes de entrada y salida sobre un sistema con pocos puntos de acceso.[3]

---

[3] Una limitación del sistema utilizado es que provee un sólo canal de comunicación con el servidor de discos.

Un punto que vale la pena mencionar con referencia a los cálculos, es que puede aprovecharse la descomposición espectral del número de onda. Es decir, los datos transformados poseen una simetría alrededor de $k=N/2+1$ tal como se puede apreciar en el algoritmo de la Fig. 1. De esta manera, basta asignar $(N/2+2)/P$ trazas sísmicas a cada procesador para efectuar la integración.

El aprovechamiento de la descomposición espectral es válido en cualesquiera de los niveles de paralelismo con el fin de reducir el cómputo. Sin embargo, sólo en el paralelismo por trazas, la descomposición espectral del número de onda contribuye a una reducción del uso de memoria local por transputador. Esto se debe a que en el paralelismo por sección los datos originales no presentan simetrías y, en el paralelismo por integración de los datos, se trabaja sobre una traza a la vez. Consecuentemente, los $4/P$ Mbytes de memoria del ejemplo se reducen a 2,16 $/P$ Mbytes. La siguiente proposición resume la reciente discusión:

Proposición 1.[4] *El orden del tiempo de cómputo en el paralelismo por trazas es el mismo al de secciones sísmicas. Sin embargo, los requerimientos de memoria de un sistema paralelo distribuido con igual capacidad, es al menos* $\frac{2}{5}P$ *veces menor en el primer caso, pero nunca* $2P$ *veces mayor en relación al segundo.*

5. Paralelismo por integración

La formulación matricial (15) puede resolverse a través de tres esquemas secuenciales: Por filas, por columnas y por bloques (Golub y van Loan [7]). El problema del producto matriz-vector en paralelo sobre diversas topologías ha sido tratado en [1], [3], [4], [6] y [8].

Cada esquema posee una misma complejidad computacional de $O(n^2)$ para una matriz de dimensión n. No obstante, la descomposición del dominio que sugiere cada uno de ellos es diferente para la paralelización del producto matriz-vector. Los dos primeros conducen a una descomposición por tiras ("stripes") unidimensionales de la matriz. El tercero en el fondo, es una combinación de los dos primeros, generándose de este modo una descomposición bidimensional (Snyder-Socha [18]).

Los requerimientos de memoria sobre un sistema paralelo varían de un esquema a otro. Sin embargo, en el peor de los casos sólo se requiere almacenar el vector, es decir, la traza sísmica a transformar. Esto se traduce en al menos 8 Mbytes de memoria global en el sistema. Como contraparte, es necesario cargar y descargar del sistema todas las trazas que conforman una sección. Más específicamente, para la ejecución del DMO sobre un sismograma completo se requieren $H*(N/2 + 2)$ cargas al sistema; por lo discutido en el paralelismo por trazas.

Explotar este nivel de paralelismo posee las siguientes implicaciones:

- Tratamiento del fragmento de la integración solamente.
- Mejor aprovechamiento de la memoria local del transputador.

---

[4] En [15] se demuestran las proposiciones y corolarios del presente trabajo.

- Diferentes posibilidades en la descomposición de la matriz.
- El manejo más sencillo de la entrada y salida en relación a los otros dos niveles de paralelismo.

En los apartados que siguen se analizan por separado cada uno de los esquemas algorítmicos planteados para la descomposición matrtiz-vector. Sin pérdida de generalidad, el análisis dejará de lado el cómputo de las entradas de la matriz . El lector podrá inferir sin mucha dificultad que el orden de este cómputo es $\frac{M^2}{P}$.

Se supone en lo que sigue, que la suma de 2 números complejos es equivalente a 2 sumas reales, y el producto de 2 números complejos es equivalente a 2 sumas y 4 productos reales. En consecuencia, se establece una relación de desempeño 1:4 entre ambas operaciones complejas, respectivamente. Esta suposición no es tan caprichosa, si se sabe que en el transputador un producto es *11/7* [12] veces más costoso que una suma de punto flotante en precisión simple (32 bits)[5]. El costo de realizar una suma o resta real en el transputador se denotará por τ.

5.1 Esquema por filas

En la Fig. 3 se muestra la descomposición de la matriz en trozos horizontales, idénticos y rectangulares. La cantidad de trozos es igual a P y cada uno es asignado a un procesador específico.

El algoritmo se puede bosquejar en tres pasos:

1. Se reparte el vector de datos de la traza y los límites verticales (i.e. el rango de filas) del trozo de fila a cada procesador.
2. En paralelo, cada nodo P_l $(l = 1, .., P)$ construye la entradas de la matriz y lo multiplica por la entrada correspondiente del vector siguiendo un esquema por filas. Si i es el índice de las filas y j el de las columnas, entonces el producto se obtiene de la siguiente manera:

$$y_i = \sum_{j=1}^{M} w_{ij} x_j \qquad i = (l-1)\frac{M}{P} + 1, .., l\frac{M}{P} \tag{7}$$

3. Se recolectan los trozos resultantes y se colocan en la posición que les corresponde dentro del vector final de resultado.

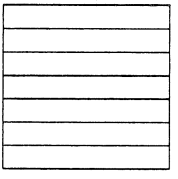


Fig. 3. Descomposición por filas

[5] Algunos autores [5], [16], han logrado maneras mas eficientes de realizar el producto de complejos sobre máquinas cuyos productos y sumas en punto flotante poseen una relación mayor de 3:1. El transputador no cae en este caso.

Proposición 2. *El tiempo T(M,P) del algoritmo paralelo por filas está dado por:*

$$\frac{5M^2}{P}\tau - \frac{M}{P}\tau + 2(P1 + P2 - 2)t_{start} + \frac{M}{2P1^2}(3P1^3 - 5P1^2 - P1 + 2PP1 + P)t_{send} \tag{8}$$

El tiempo de cómputo del algoritmo secuencial *T(M,1)* está dado por $5M^2\tau - M\tau$. Por lo que si se desprecia el tiempo de comunicación, el aceleramiento será lineal con respecto al número de procesadores.

5.2 Esquema por columnas

El algoritmo de columnas difiere levemente del de filas en los siguientes puntos:

1. Cada procesador $P_l \quad (l = 1, .., P)$ recibe el trozo del vector columna que va desde la entrada con índice $(l-1)\frac{M}{P} + 1$ hasta la entrada $l\frac{M}{P}$.
2. El producto en cada procesador se efectúa como sigue:

$$y_i^l = \sum_{i=1}^{M} w_{ij}x_j \qquad j = (l-1)\frac{M}{P} + 1, .., l\frac{M}{P} \tag{9}$$

   Esto es, y^l se escribe como combinación lineal de las $\frac{M}{P}$ columnas que le fueron asignadas, por lo que su longitud es M.
3. Se recolectan y suman entre sí los vectores y^l resultantes de cada procesador P_l

A pesar de que la comunicación es la misma en ambos algoritmos, el cómputo no lo es. Esto se debe primordialmente al paso 3. Nótese que los vectores obtenidos en cada procesador son resultados parciales del vector final de resultado. Estos son sumados a medida que se recolectan los datos. Más específicamente:

$$y = \sum_{j=1}^{M} y_j^l \qquad l = 1, .., P \tag{10}$$

Esta suma ocurre en forma encadenada y sólo en paralelo a nivel de las filas o columnas de la malla del procesador. La siguiente proposición resume esta discusión:

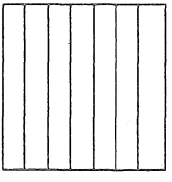


Fig. 4. Descomposición por columnas

Proposición 3. *El tiempo T(M,P) del algoritmo paralelo por columnas está dado por:*

$$\frac{5M^2}{P}\tau + (P1 + P2 - 3)M\tau + 2(P1 + P2 - 2)t_{start} + \frac{M}{2P1^2}(3P1^3 - 5P1^2 - P1 + 2PP1 + P)Mt_{send} \tag{11}$$

A diferencia del algoritmo por filas, el de columnas no realiza todas las operaciones (i.e. sumas) en paralelo. En el algoritmo anterior cada trozo resultante es asignado a la posición que le corresponde en el vector resultante. El identificador de cada procesador y el número de filas repartidas es suficiente para lograr dicha asignación.

Corolario 1. *Si la malla de procesadores es cuadrada (i.e. $P1 = P2 = \sqrt{P}$) entonces el tiempo óptimo de cómputo del algoritmo por columnas ocurre cuando:*

$$P^* = \sqrt[3]{25M^2} \approx 2,92M^{\frac{2}{3}} \tag{12}$$

Más aún, el tiempo óptimo de cómputo del algoritmo es de orden $O\left(M^{\frac{4}{3}}\right)$

Para tener una mejor idea de lo que significa el resultado del corolario, se puede tomar por ejemplo una matriz de dimensión 512. Entonces el requerimiento óptimo de procesadores es

$$P^* = \sqrt[3]{25 * (512)^2} \approx 187,14 \tag{13}$$

En el mejor de los casos, esto implica contar con una malla cuadrada de 14x14.

Si se sustituye (12) en las ecuaciones (8) y (11), se realiza el cociente y se toma el límite cuando M crece al infinito, resulta:

$$\lim_{M\to\infty} \frac{T(M,P) \text{ por filas}}{T(M,P) \text{ por columnas}} = \frac{5}{5 + 2 * 5^{\frac{1}{6}}} \approx 0,6565 \tag{14}$$

Asintóticamente, el algoritmo por filas es aproximadamente uno y medio veces *(1,52)* más rápido que el de columna. En este cálculo simplificado se está suponiendo una cantidad infinita de memoria por procesador y un ancho de banda infinita por canal.

5.3 Esquema por bloques

La idea básica del algoritmo por bloques es descomponer la matriz en $P1 \times P2$ submatrices de tamaño $\frac{M}{P1} \times \frac{M}{P2}$. De manera formal, el producto matriz-vector se replantea de la siguiente forma:

$$y_i^\mu = \sum_{\nu,j} W_{i,j}^{\mu,\nu} x_j^\nu \qquad 1 \le \mu \le P1\,, \quad 1 \le \nu \le P2 \tag{15}$$

donde μ y ν son los subíndices de los bloques.

Los vectores y^{μ} y x^{ν} son de longitud $\frac{M}{P1}$ y $\frac{M}{P2}$ que se obtienen de dividir y y x entre $P1$ y $P2$, respectivamente. Es decir,

$$x_{\frac{M}{P2}(\nu-1)+j} = x_j^{\nu}$$

$$y_{\frac{M}{P1}(\mu-1)+i} = y_i^{\mu} \tag{16}$$

Suponiendo que cada procesador de la malla está etiquetado por el par (μ, ν), el algoritmo se presenta como sigue:

1. Se reparten los elementos de x^{ν} y los límites verticales y horizontales que definen la matriz-bloque $W^{\mu\nu}$ a cada procesador $P_{\mu,\nu}$.
2. Se efectúa en paralclo, el producto matriz-vector para los bloques. Cada bloque (μ, ν) utiliza un esquema secuencial por filas[6]:

$$y^{\mu(\nu)} = \sum_{j=1}^{M/P2} W_{i,j}^{\mu,\nu} x_j^{\nu} \tag{17}$$

   sin realizar sumas sobre el índice ν.
3. Se recolectan y suman los resultados parciales. Esto se puede efectuar de dos maneras:

   a. Recolectar y sumar horizontalmente los resultados $y^{\mu(\nu)}$. Es decir:

$$y^{\mu} = \sum_{\nu=1}^{M/P2} y^{\mu(\nu)} \tag{18}$$

   Luego en sentido vertical, se colocan los trozos y^{μ} que están distribuidos en la primera columna de la malla en un sólo vector final resultado en el procesador raíz (i.e. el procesador que está conectado con el servidor o anfitrión). Este criterio se denominará suma-asignación.

   b. Recolectar los resultados parciales en un vector de longitud M. De esta manera, los resultados quedan localizados sobre la primera fila de la malla de procesadores. Luego, estos se suman horizontalmente en dirección del nodo raíz que al final poseerá el vector final resultado. Esta suma se describe analíticamente como:

$$y^{\mu} = \sum_{\nu=1}^{M} y^{\mu(\nu)} \tag{19}$$

   El criterio se denominará asignación-suma.

---

[6] También es igualmente válido tomar el algoritmo por columnas.

Proposición 4. *El tiempo T(M,P) del algoritmo paralelo por bloques usando el criterio suma-asignación está dado por:*

$$\frac{5M^2}{P}\tau + (P2-2)\frac{M}{P1}\tau + 2(P1+P2-2)t_{start} + \frac{M}{2PP1^2}(P1^5 + P1^4 - 4P1^3 - 2P1P + 2PP1^2 + 2P^2)t_{send} \tag{20}$$

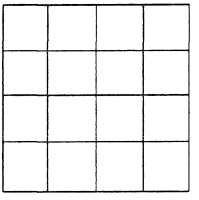


Fig. 5. Descomposición por bloques

Proposición 5. *El tiempo T(M,P) del algoritmo paralelo por bloques usando el criterio asignación-suma está dado por:*

$$\frac{5M^2}{P}\tau + (P2-1)M\tau - \frac{M}{P1}\tau + 2(P1+P2-2)t_{start} + (2P^2 - PP1 + 2P1^3 + PP1^2 - 4P1^2)\frac{M}{PP1}t_{send} \tag{21}$$

De las dos proposiciones anteriores, se ve claramente que el criterio suma-asignación reduce tanto el tiempo de cómputo como el de comunicación cuando $P1 > 1$. La explicación es la misma que para el caso columna: El criterio suma-asignación aprovecha mejor el paralelismo para la suma de vectores e implica el manejo de vectores más pequeños en la transmisión de datos.

Comparando las expresiones (8), (11) y (20) se infiere que los tiempos de cómputo ordenados de menor a mayor son:

$$T_{fila} < T_{bloque} < T_{columna} \tag{22}$$

Ahora, ¿hasta qué punto es bueno el algoritmo por bloques?. La inquietud se traduce en analizar cuál es la configuración de bloques y malla de procesadores más idónea que arroja los menores tiempos de cómputo. Para ello, se reduce la componente $(P2-2)\frac{M}{P1}$ de la expresión (20). Es fácil entrever que $P2$ debe disminuir y $P1$ aumentar para lograr ese objetivo. De

esta manera, el cálculo por bloques se reduce a una descomposición por bloques de filas de la matriz de DMO.

6. Resultados experimentales

La implantación de los programas se realizó sobre una plataforma de transputadores usando el lenguaje de programación Occam [17] y el ambiente Toolset [13].

6.1 Filas

La Tabla 1 condensa los resultados obtenidos para la descomposición homogénea por filas de la matriz de DMO.

Dimension de la matriz	8 transputadores	4 transputadores	2 transputadores	1 transputador
256	0,4768	0,9424	1,8798	3,7557
512	1,8995	3,7789	7,5535	15,0847
1024	7,5868	15,1383	30,2594	60,4737
2048	30,3281	60,5794	121,1067	242,8271
4096	121,2524	242,3346	484,5697	970,4310

Tabla 1 Tiempos de ejecución (en segundos) para el algoritmo paralelo de filas.

Si se analiza la Tabla 1 en sentido vertical, se observa que los tiempos se cuadruplican aproximadamente al duplicar la dimensión de los datos. Esto era de esperarse por la misma ecuación (8) pues si se duplica el tamaño del problema, esto es, se pasa de M a $\tilde{M} = 2M$, se obtiene que :

$$T\left(\tilde{M}, P\right) \simeq \frac{5\tilde{M}^2}{P} = \frac{5(2M)^2}{P} = \frac{5\left(4M^2\right)}{P} \simeq 4T(M, P) \tag{23}$$

Nótese que el tiempo de comunicación no incidió decisivamente en el tiempo global del algoritmo paralelo.Een la Tabla 2 se aprecian los correspondientes valores de aceleramiento.

Dimensión de la matriz	8 transputadores	4 transputadores	2 transputadores	1 transputador
256	7,8768	3,9852	1,9979	1,0
512	7,9414	3,9918	1,9970	1,0
1024	7,9709	3,9947	1,9985	1,0
2048	8,0066	4,0084	2,0050	1,0
4096	8,0033	4,0045	2,0026	1,0

Tabla 2 Aceleramientos del algoritmo paralelo por filas

No obstante, de dicha tabla se pueden extraer muchas ideas interesantes. Primero, la tabla muestra en sentido horizontal los cocientes de tiempos para un tamaño de problema fijo y usando diferente número de procesadores. Para problemas más pequeños se observa un descenso más acentuado del aceleramiento a medida que se aumentan el número de procesadores. Esto se debe a la incidencia de dos factores fundamentalmente: El tiempo de comunicación y el tiempo de carga del vector.

Obsérvese un aumento del aceleramiento en el sentido vertical de la Tabla 2. Es más, el aceleramiento alcanza valores supralineales pues excede el número de procesadores empleados.

Gustavson [9] menciona una de las causas fundamentales para que esto ocurra: el aprovechamiento de la memoria. En el transputador existen dos niveles de memoria [2]:

- Dentro del "chip", con una capacidad de 4 Kbytes (i.e. 1024 valores de punto flotante).
- Fuera del "chip", que reside en la tarjeta que soporta el transputador y posee una capacidad de 1 Mbyte (i.e. 524.288 valores de punto flotante).

En el algoritmo por fila (y similarmente en los demás algoritmos), se usan dos vectores[7] de longitud M y dos de M/P para guardar el resultado. Como el programa reside en la parte más alta de la memoria (i.e. luego de todas la variables declaradas), todos los vectores permanecen dentro del "chip" si:

$$M < 512\frac{P}{P+1} \tag{24}$$

Una particularidad del compilador de Occam es que coloca las últimas variables declaradas en las localidades más bajas de la memoria [2]. Esto quiere decir, que las variables declaradas de primero corren el riesgo de quedar fuera de la memoria del "chip" ya que ésta ha sido ocupada las variables declaradas posteriormente. En la codificación de los programas, se declararon de último los vectores resultados.

A efectos prácticos, se puede suponer el caso borde que para $M = 512\frac{P}{P+1}$ no se sale ningún valor de la memoria dentro del "chip". Haciendo esta salvedad y usando como base la desigualdad (24), se puede construir la Tabla 3.

La Tabla 3 muestra que si la matriz posee una dimensión menor de 512, los tiempos de acceso a memoria son los mismos, pero al aumentar de esa cantidad comienzan a aparecer irregularidades en el tiempo de ejecución sobre el sistema de memoria distribuido. Esto se evidencia, si por ejemplo se estima el tiempo de ejecución para la dimensión del problema igual a 1024. Como ilustración se puede considerar que la operación que gobierna el cómputo es el acceso a memoria. Así, el tiempo de ejecución secuencial se desglosa como 1024 accesos a la memoria del "chip" y 3072 fuera de él. Por tanto, si τ_m es el costo de un acceso a memoria:

$$T(M,1) \approx 10 * 1024^2\tau_m + 6 * 1024\tau_m \tag{25}$$

[7] Uno para representar la parte real y el otro la parte imaginaria del complejo.

Procesa-dores	Dimensión 256	Dimension 512	Dimensión 1024	Dimensión 2048	Dimensión 4096
1	0	1024	3072	7168	15360
2	0	512	2048	5120	11240
4	0	256	1536	4096	9216
8	0	128	1280	3584	8192

Tabla 3. Cantidad de valores que quedan fuera de la memoria del "chip".

El término elevado al cuadrado se refiere al acceso del vector columna dentro del ciclo más interno de la integración, y el lineal al acceso del vector resultado en el lazo más externo. Esto se debe a que la suma de los términos que resultan del producto interno fila de la matriz con el vector columna se acumulan en una variable para luego ser asignadas en el lazo más inmediatamente exterior en la entrada correspondiente del vector resultado. De allí que la intensidad de los accesos a memoria estén regidas por el vector columna.

Si se usan dos transputadores la cifra se convierte en:

$$T(M,2) \approx 10 * \frac{1024^2}{2}\tau_m + 2 * 512\tau_m \tag{26}$$

Entonces, el correspondiente aceleramiento para dos procesadores viene dado por:

$$A(M,2) = \frac{T(M,1)}{T(M,2)} = \frac{20 * 1024 + 12}{10 * 1024 + 2} \approx 2,0008 \tag{27}$$

De esta manera se ilustra la procedencia de este acceleramiento supralineal. En general, la expresión de aceleramiento para P procesadores y un problema de tamaño $M \geq 1024$ es:

$$A(M,P) = \frac{10M^2 + 10M - 4096}{10M^2 + 10M - 4096P}P \tag{28}$$

Claramente, al aumentar el número de procesadores el coeficiente que acompaña a P aumenta también.

Aparentemente, la contribución de las diferencias de tiempo para el acceso a memoria se suman a un incremento del tiempo de comunicación, aunque sigue predominando el cómputo asociado a la integración (i.e. divisiones, productos, seno, coseno, etc.). De hecho si se incrementa el tamaño del problema, la expresión (28) disminuye para un número fijo de procesadores P, confirmándose la atenuación de los efectos de memoria.

El concepto de fracción serial aporta aún más información sobre el aprovechamiento de los recursos computacionales por parte del algoritmo. La Tabla 4, indica que la máxima fracción serial está asociada con el problema de dimensión 32: 0,0227, esto es, 2,27 % del algoritmo paralelo. Ahora bien, se presentan irregularidades que valen destacar de la tabla. La columna asociada al uso de 4 transputadores presenta valores más pequeños que en las columnas restantes. Esto levanta la sospecha que al contar con una malla cuadrada, el "overhead" de comunicación resulta menor que en las mallas rectangulares de 2x1 y 2x4 procesadores. Si bien por la falta de procesadores esta situación no es confirmable, es claro que el tiempo de t_{start} en la ecuación (8) alcanza su menor valor cuando la malla es cuadrada. Lo mismo no se desprende de manera tan evidente para t_{send} [15].

Dimensión de la matriz	8 transputadores	4 transputadores	2 transputadores	1 transputador
256	0,0022	0,0012	0,0011	1,0
512	0,0010	0,0006	0,0015	1,0
1024	0,0005	0,0004	0,0007	1,0
2048	0.0000	0.0000	0.0000	1,0
4096	0.0000	0.0000	0.0000	1,0

Tabla 4 Fracción serial ∫ del algoritmo por filas.

Los valores de ∫ reflejan sin embargo, que a medida que crece el problema en el procesador, disminuye la fracción serial. Esto se debe a que crece la fracción paralela asociada al cómputo.

6.2 Columnas

En el algoritmo por columnas se obtuvieron los siguientes tiempos:

Dimensión de la matriz	8 transputadores	4 transputadores	2 transputadores	1 transputador
256	0,4926	0,9667	1,9250	3,8077
512	1,9568	3,8814	7,7477	15,4189
1024	7,8085	15,5360	31,1084	62,2270
2048	31,1464	62,2894	124,6848	248,4312
4096	124,9169	249,9302	500,0774	997,1921

Tabla 5 Tiempos de ejecución (en segundos) para el algoritmo paralelo por columnas.

Los valores de tiempo son mayores a los del algoritmo por filas, tal como era de esperarse según el desarrollo analítico expuesto en el capítulo anterior. Igualmente se encontró la presencia de valores supralineales que pueden ser explicados de manera análoga al caso de filas [15].

Además, se obtuvo el mismo comportamiento en cuanto a la fracción serial, un decremento relativo para el uso de 4 transputadores con respecto al uso de 2 y 8 transputadores.

6.3 Bloques

En el algoritmo por bloques, cada procesador se encargó de ejecutar en forma local el algoritmo por filas. El esquema utilizado fue de suma-asignación. De esta manera, para el uso 1 y 2 transputadores se llega al caso particular del algoritmo paralelo por filas. Por lo tanto, las siguientes tablas hacen sólo mención a las mallas de 2x2 y 2x4 procesadores.

Dimension de la matriz	8 transputadores	4 transputadores
256	0,4815	0,9501
512	1,9109	3,7956
1024	7,6152	15,2104
2048	30,4692	60,9308
4096	121,2524	243,8778

Tabla 6 Tiempos de ejecución (en segundos) para el algoritmo paralelo por bloques.

Se obtuvieron tiempos menores a los del algoritmo por columnas pero mayores a los logrados con el algoritmo por filas tal como lo sugiere (22). Las diferencias se hacen más notables a medida que aumenta la dimensión del problema. Al observar los aceleramientos se encuentró el mismo patrón de comportamiento. Las fracciones seriales dejan entrever la misma situación anterior.

6.4 Comparación de los diferentes esquemas paralelos

La Tabla 7 resume los tiempos de ejecución usando el máximo potencial del sistema para los algoritmos propuestos. Es evidente que los tiempos del algoritmo por bloques se aproximan más a los del algoritmo por filas que al de columnas. Esto también se puede concluir mediante los tiempos de cómputo de la ecuaciones (8), (11) y (20).

Si se efectúan los cocientes columna-bloque y bloque-fila se obtienen respectivamente:

$$\frac{1+\frac{24}{5M}}{1+\frac{8}{5M}} \quad \text{y} \quad \frac{1+\frac{8}{5M}}{1-\frac{1}{5M}} \tag{29}$$

Comparando ambos cocientes, se obtiene que $M > 2{,}5$ lo que se cumple trivialmente.

Dimension de la matriz	Fila	Bloque	Columna
256	0,4768	0,4815	0,4926
512	1,8995	1,9109	1,9568
1024	7,5868	7,6152	7,8085
2048	30,3281	30,4692	31,1464
4096	121,2524	121,9490	124,9169

Tabla 7 Comparación entre los tiempos (en segundos) de las diferentes descomposiciones homogéneas sobre 8 transputadores..

Adicionalmente, se puede determinar cuán cerca están los tiempos del algoritmo por bloques con respecto a los otros dos. Para ello, se escribe éste como una combinación lineal de los tiempos de cómputo del algoritmo por filas y por columnas. En otras palabras:

$$\rho\left(\frac{5M^2}{P} - \frac{M}{P}\right)\tau + (1-\rho)\left(\frac{5M^2}{P} - (P1+P2-3)M\right)\tau = \frac{5M^2}{P} + M \tag{30}$$

donde $0 < \rho < 1$. Por lo que:

$$\rho = \frac{P(P1+P2-4)}{P(P1+P2-3)+1} \tag{31}$$

Para $P1$=2 y $P2$=4, esto indica que el tiempo del algoritmo por bloques está a 25/16 del tiempo del algoritmo por filas y en consecuencia, a 25/9 del algoritmo por columnas.

Si se hacen los cálculos en base a los valores de la tabla, como por ejemplo los tiempos asociados al problema 512 se tiene que:

$$1,8995\rho + (1-\rho)1,9568 = 1,9109 \quad \Rightarrow \quad \rho = \frac{0,0459}{0,0573} \approx 0,8 \tag{32}$$

Este valor se aleja en un 16% del valor estimado por (31), probablemente como resultado de omitir la influencia del tiempo de comunicación y la memoria.

6.5 Desempeño de otras arquitecturas

Con el fin de medir el alcance de los tiempos obtenidos, se comparó el desempeño de los mejores algoritmos con otros computadores. Para efectos de esta comparación se usó el computador IBM 3090 con y sin facilidad vectorial, la máquina RS/6000 modelo 530 de IBM

que es conceptuada como la estación más rápida que hay en el mercado actualmente y dos modelos diferentes de las estaciones de trabajo SUN. Los resultados se muestran en la Tabla 8 De ella se puede deducir la siguiente relación jerárquica a partir de los tiempos:

Dimen-sión	IBM 3090/VF	8T por filas	IBM 3090	RS6000-530	SUN 4/ 260	SUN SPARC-1
512	0,59	1,89	1,90	3,58	6,20	12,50
1024	2,21	7,58	7,50	14,28	24,80	51,00
2048	8,66	30,33	29,90	56,94	98,80	202,50
4096	34,46	121,25	119,26	228,23	395,60	822,30

Tabla 8 Tiempos (en segundos) para la integración de una traza particular sobre diferentes arquitecturas.

De acuerdo con la derivación analítica del algoritmo por filas, y a lo que se puede observar en la tabla, se podría alcanzar el desempeño en la IBM3090/VF usando aproximadamente el triple del número de transputadores y triplicando el tamaño del problema para mantener la eficiencia [15].

6.6 Comparación entre el paralelismo por trazas y paralelismo por integración

Se puede observar en la Tabla 9 que los resultados son muy similares, aunque son levemente menores en el caso del paralelismo por integración de los datos. Esta diferencia puede atribuirse a las diferencia de tiempo de acceso a memoria, ya que el paralelismo por trazas involucra más accesos de memoria fuera del "chip".

Dimensión de la sección	Integración de los datos usando el esquema por filas	Paralelización por trazas
64x128	7,7568	7,8985
64x256	30,5270	31,4800
64x512	121,6204	125,8772
62x1024	486,6712	503,8840

Tabla 9 Tiempos (en segundos) del paralelismo por trazas e integración de los datos sobre 8 transputadores.

7. Conclusiones y perspectivas

Se concluyen los siguientes aspectos en torno al trabajo:

1. El DMO es una aplicación en la cual es factible explotar los tres niveles de paralelismo estudiados.
2. La demora por tiempo de comunicación es realmente ínfima en relación a los tiempo de cómputo.
3. Se mostró una gran semblanza entre los resultados teóricos y empíricos.
4. De los resultados obtenidos, se puede construir una jerarquía en base a los tiempos de respuesta de cada algoritmo implantado para el nivel de integración de los datos:
   a. Por filas
   b. Por bloques
   c. Por columnas

   Si se tuviese que ubicar el paralelismo por trazas, con el fin de obtener una jerarquía más general, éste posiblemente ocuparía la última posición, según se desprende de la Tabla 9.
5. A través de los resultados empíricos, se pudo observar la enorme incidencia de los accesos a memoria dentro o fuera del "chip" del transputador.
6. La fracción serial ciertamente aporta una información práctica que no es detectada con las demás métricas. En los experimentos realizados, mostró indicios de las posibles anomalías que se pueden presentar al trabajar sobre una malla cuadrada o rectangular de procesadores.
7. Existen ciertas limitaciones de memoria y capacidad de comunicación en la plataforma la arquitectura utilizada para explotar cada uno de los niveles de paralelismo encontrados en el DMO.

Se recomiendan las siguientes acciones:

1. Estudiar con mayor detenimiento el problema de entradas repetidas en la matriz que se derivan del operador para reducir la cantidad de cálculo.(El estudio de este problema se está tratando actualmente y su publicación está en curso).
2. Explorar el paralelismo por secciones y por trazas sísmicas en mayor detalle.
3. Conjugar los tres niveles de paralelismo con el fin de explotar un mayor grado de paralelismo.
4. Llevar las experiencias computacionales a un mayor número de procesadores y tamaño del problema.

Referencias

[1] Akl, S. *The Design and Analysis of Parallel Algorithms.* Englewood Cliffs, N. J., Prentice Hall, 1989.

[2] Atkin, P. *Performance Maximisation.* INMOS Technical Note. Bristol, N. 17: 1-26.

[3] Codenotti, B. y Puglisi, C. Matrix-Vector Multiplication: Parallel Algorithms and Arquitectures. *Computer Mathematical Applications.* 12(16): 1057-1063, 1988.

[4] Codenotti, B., Lotti, G. y Romani, F. Area-Time Trade-Offs for Matrix-Vector Multiplication. *Journal of Parallel and Distributed Computing.* 1(8): 52-59, 1990.

[5] Fam, A. Efficient Complex Matrix Multiplication. *IEEE Transactions on Computers.* 7(C-37): 877-879, 1988.

[6] Fox, G. et al. *Solving Problems on Concurrent Computers.* Englewood Clifffs, N.J. Prentice Hall, 1988.

[7] Golub, G. y van Loan, Ch. *Matrix Computations* . Johns Hopkins University Press, 2nd edition, 1989.

[8] Grigg, K., Miguet, S. y Robert, Y. Complexity of the Symmetric Matriz-Vector Product on a Ring of Processors. *1990, Fifth Distributed Memory Computing Conference.* South Carolina, Vol II: 1324-1333.

[9] Gustavson, J. Fixed Time, Tiered Memory and Superlinear Speedup *1990, Fifth Distributed Memory Computing Conference.* South Carolina, Vol II: 1255–1260.

[10] Hale, D., Dip-moveout by Fourier Transform. *Geophysics*, 6(49): 741–757, 1984

[11] Hale, D., *Dip-moveout Processing : Course Notes.* . Colorado School of Mines, Tulsa, 1988.

[12] INMOS. *The Transputer Databook.* 2nd Edition, 1989.

[13] INMOS. *Occam 2 Toolset.* User manual, 1989.

[14] Karp A., y Flatt, P. Measuring Parallel Processor Performance. *Communications of the ACM.* 5(33): 539-543, 1990.

[15] Klíe, H. *Descomposición de Dominios: Caso de Estudio.* Tesis de Maestría, Universidad Simón Bolívar, Febrero 1991.

[16] Moharir, P. Extending the Scope of Golub's Method Beyond Complex Multiplication. *IEEE Transaction on Computers.* 5(C-34): 484-487, 1985.

[17] Pountain, D. y May D. *A Tutorial Introduction to Occam Programming.* INMOS, 1988.

[18] Snyder, L. y Socha D. An Algorithm Producing Balanced Partitionings of Data Arrays. *1990, Fifth Distributed Memory Computing Conference.* Vol II: 867-874.

MULTIS: A Knowledge Acquisition System Based on Problem-Solving Primitives

Yuri Adrian Tijerino
Tadahiro Kitahashi
Riichiro Mizoguchi

Institute of Scientific and Industrial Research,
Osaka University

Abstract

Building knowledge-based systems is not an easy task, especially when it comes to acquiring and coding expertise from human experts into machine understandable forms. Attempts, some successful, have been made to automate this very tedious task. Nevertheless, those efforts have been focused on specific task areas of knowledge-rich human problem solving, e.g. diagnosis, classification, etc. So far, few automated knowledge acquisition systems exist that are useful in more that one such area. The aim of this project is to develop a method for automating knowledge acquisition that can be employed to automatically generate knowledge-based systems making few initial constraints on the type of problem solving mechanism. This method is based on the following concepts: (1) human problem solving can be broken down into conceptual primitives that can be reused and rearranged to model problem solving processes from vast domains and task types, and (2) there exist enough formal problem solving mechanisms in algorithmic form, or at least those mechanism can be easily developed by computer experts, to implement those models in a machine understandable code. To prove the tractability of these concepts we are currently developing MULTIS (MULti-Task Interview System).

Keywords: Interview system, cognitive primitive, generic vocabulary, generic process, building block, case-based reasoning.

1. Introduction

In the past two decades intensive research has taken place in the area of knowledge-based system in numerous countries. As a result, so-called "expert systems" have been constructed to store human problem solving knowledge in the computer in a dynamic form that can be used as an aid to the knowledge provider and other parties interested. Originally, those systems, which will be referred to as knowledge-based systems (KBS) hereof, were computer programs designed by "knowledge engineers," a term coined by Prof. Faigenbaum in the early years of artificial intelligence. Knowledge engineers are usually persons that would spend numerous hours talking to human experts, who in some cases would think of them as intruders poking in their own business. Through this painful but often rewarding period, knowledge engineers have accumulated readily useful knowledge of their own that has been written in the form of papers, books, etc. This knowledge of how to acquire knowledge from experts and encode it into computer programs, can now serve another very important purpose, that is, to create systems based on that knowledge to automate the process of knowledge acquisition.

Interview systems are computer programs that are constructed with that purpose in mind. Early interview systems were generated from existing knowledge-based systems and were useful in aiding in the construction of other knowledge-based systems that presented problem solving mechanisms similar to that of the original one. Based on this fact, it was proposed that there exist cognitive primitives (Chandrasekran, 1986; Clancey, 1985; McDermott, 1988, Tijerino et al., 1990) in which terms most human problem solving could be modeled. But how could that be possible for all facets of problems? Problem solving knowledge was subdivided in two major areas: task knowledge and domain knowledge. From this perspective, primitives could then be identified for task knowledge. Since it was also proposed by the above researchers that task knowledge presented strong guidance in the process of acquisition of domain knowledge, then those primitives would be of great help in that process too.

The aim of this project is to develop an interview system, MULTIS, based on the concept of cognitive primitives, also known as mediating representations. The major difference between this project and other related ongoing projects is that we make a distinction between domain expert's cognitive primitives and knowledge engineering primitives. Other projects consider the only the latter leaving it up to the knowledge engineer to analyze the domain expert's problem solving process and synthesize KBSs with reusable mechanisms. We propose that the former can be extremely useful in the process of modeling problem solving interactively with the domain expert and that the former can be mapped to the latter, thus automatically generating the target KBS.

2. Two-Level Domain Expert's Cognitive Primitves with different Grain Size

We make no claim that we have identified "the" primitives of human thought. On the contrary, in this paper an experimental framework is presented in which conceptual primitives do not necessarily have to be precise in modeling human thought, but only accurate in creating systems that model thought processes in the form of computer programs.

In this section, we will present two types cognitive primitives used to model expert problem solving processes: generic vocabulary and generic processes. Notice that these are only equivalent to domain expert's cognitive primitives and no attempt is made to implement them directly as computer programs. As figure 2-1 illustrates, their implementation is accomplished by mapping them to mechanisms that represent knowledge engineer's primitives which we refer to as building blocks and will introduce in another section. A comparison of our research with other ongoing research efforts will also be given. Last, we will provide an example of how an already existent system can be modeled with those primitives.

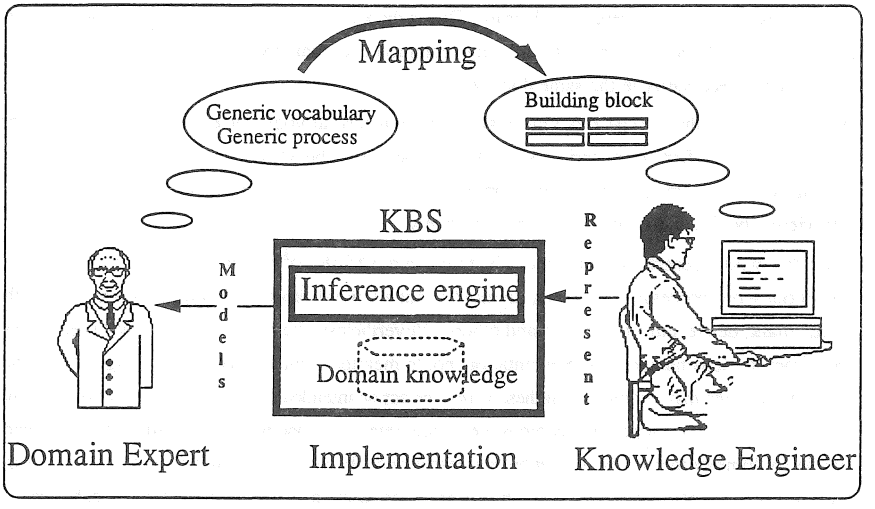


Figure 2-1. Implementation of a KBS is accomplished by mapping domain expert's primitves to knowledge engineer's primitives.

2.1 Generic Vocabulary

One kind of cognitive primitive used by all fellow human beings can be identified as the vocabulary. Not a particular language itself, e.g. English, Japanese or Spanish, but the concept represented by the vocabulary of the language, e.g. the concept given by the word "book." It is important to make this distinction because it is what makes communication possible or impossible between two parties. For instance, let's suppose a person A does not know the meaning of the word "book." Let's suppose further that another person B utters the word "book" to A in A's language. Tough A will not understand the word, it does not imply A's ignorance of the word itself, e.g., by hearing the word A has just learned of its existence. On the other hand, A will not learn the meaning of the word, i.e. the concept implied by the word, until either B shows a book or explains its meaning to A. Furthermore, once the assimilation process has been completed, A will add the new concept given by the word to his/her vocabulary, not his/her language, and will be able to use it or understand it in future occasions.

Vocabulary can either be very generalized or very specialized. Generalized vocabulary is learned at the period of adaptation during one's youth and as one becomes older or experienced at doing some particular task or job one learns more specialized vocabulary. The latter is usually explained in terms

of the former, which is not a reciprocal relation. For instance, the concept of book can be explained in terms of words such as paper, writing, tales, drawing, etc, but those words cannot be explained in terms of book without major effort. As such vocabulary becomes more and more specialized, it becomes dependent on some particular task and common only to individuals with experience in that task. Furthermore, as one gains a vocabulary related to an specific task in which one is experienced, e.g. medical or equipment failure diagnosis, that vocabulary becomes the basis for explaining concepts more specialized and for communicating with colleagues, in common task-dependent vocabulary, without resorting to more generalized terms. This is the kind of vocabulary to which we will refer to as "generic vocabulary." Generic because it is common to domain experts performing the same task type, but not necessarily in the same domain.

We assume that there exist this kind task-dependent terminology for each PS task that can be represented in the form of generic vocabulary. Generic vocabulary can serve two purposes, that is, to analyze the tasks for hidden cognitive primitives, which we call generic processes* and to help the interviewer be fluent and coherent during interview.

In the real world, generic vocabulary can be found in most fields of human cognitive processes. Let's revisit our word-processing example. In most, if not all, word-processing programs one can use the words mark, delete, cut, copy, paste and others. Nevertheless, the name of this actions (verbs) may vary from implementation to implementation, not to mention from language to language. No matter how different one may choose to call these verbs, the action intended is still very similar if not the same. Due to this fact, when a person becomes well acquainted with one word processor it becomes very simple for him/her to use any other word processor once the necessary mapping or translation of the previous word-processor's vocabulary has been made to the new word-processor's vocabulary.

In this early stages of our project, we have identified a generic vocabulary for the task of scheduling. In table 2-1, the vocabulary we have identified for scheduling is depicted. The words in *italics* represent verbs or actions which we also call active vocabulary. The rest are called passive vocabulary because usually actions are performed on them. The passive vocabulary is divided into two types: those with strong task dependance, represented in outline font, and those with weak task dependance, represented in normal font.

2.2 Generic Processes

Generic processes are another kind of mediating representation between the domain expert's cognitive primitives and knowledge engineering primitives. Their main difference with generic vocabulary is that generic vocabulary are mostly partial concepts with no significant meaning unless combined with others and generic processes are primitive concepts of activity described in terms of generic vocabulary. Generic vocabulary are passive in the sense that though some of them represent an action, they must be combined with other vocabulary to completely describe the action, e.g. what is the object of the action. As the terms mediating representation implies, generic processes are not "the" cognitive primitives of human thought and no attempt is made to categorize them as such. The generic processes presented here for scheduling are experimental processes that have been changing since first introduced in order to better and more generally represent the processes undertaken in scheduling. They stand some where in the spectrum between real domain expert's cognitive primitives and knowledge engineer's building blocks.

Table 2-1. Experimental generic vocabulary for scheduling.

analyze	*delete*	*relax*
apply	*generate*	requirement
assign	*identify*	resource
batch	job	*satisfy*
capacity	*merge*	*select*
categorize	*modify*	sequence
category	preference	schedule
conflict	priority	subschedule
constraint	process	tolerance
		update

Italics : action Normal: weak task-dependence

Outline: strong task dependencce

Generic processes are primitives defined in terms of generic vocabulary. In our word-processing example, a generic process could be paragraph-deletion for which a paragraph has to be marked before being deleted. The generic process to generic vocabulary relation can be written as: paragraph-deletion = delete marked paragraph. The most important relation between generic vocabulary and generic processes is that generic processes usually combine two or more generic words to describe a general action, not another word.

We have identified some generic processes for planning and scheduling, but intend to expand to other PS tasks such as monitoring, diagnosis and so forth. Although, perfect generic vocabulary and generic process libraries cannot be constructed, we believe that they provide enough guidance for analysis and synthesis of the tasks under consideration. What matters the most, is that the mechanisms, to which the generic processes were mapped, are able to successfully accomplish their final goal, that is, that they will help construct (synthesize) the desired KBS. In MULTIS, a mapping between generic processes and knowledge engineer's mechanisms is made in such a way that one or more generic processes may correspond to a single mechanism.

Figure 2-2 presents some generic processes identified for scheduling and simple planning problems. We have divided the generic processes in two levels: generic processes and generic sub-processes. The former are more specific to the task and the later more general, that is, the later are more general and can be used to explain the process of the former. In this figure, the left column depicts the generic processes and the right columns the generic sub-processes. The lines between columns indicate modularity or specialization. Appendix A presents a more detailed explanation for each of the processes shown in figure 2-2.

Generic processes can be interconnected in what we refer to as generic process networks. The connectability between generic processes is restricted by the kind of conceptual input or output, represented in the form of generic vocabulary, that they can receive or send respectively. Thanks to this connectability relation, inference strategies can be created to aid in the process of identifying what generic processes a particular domain expert employs when performing a specific task.

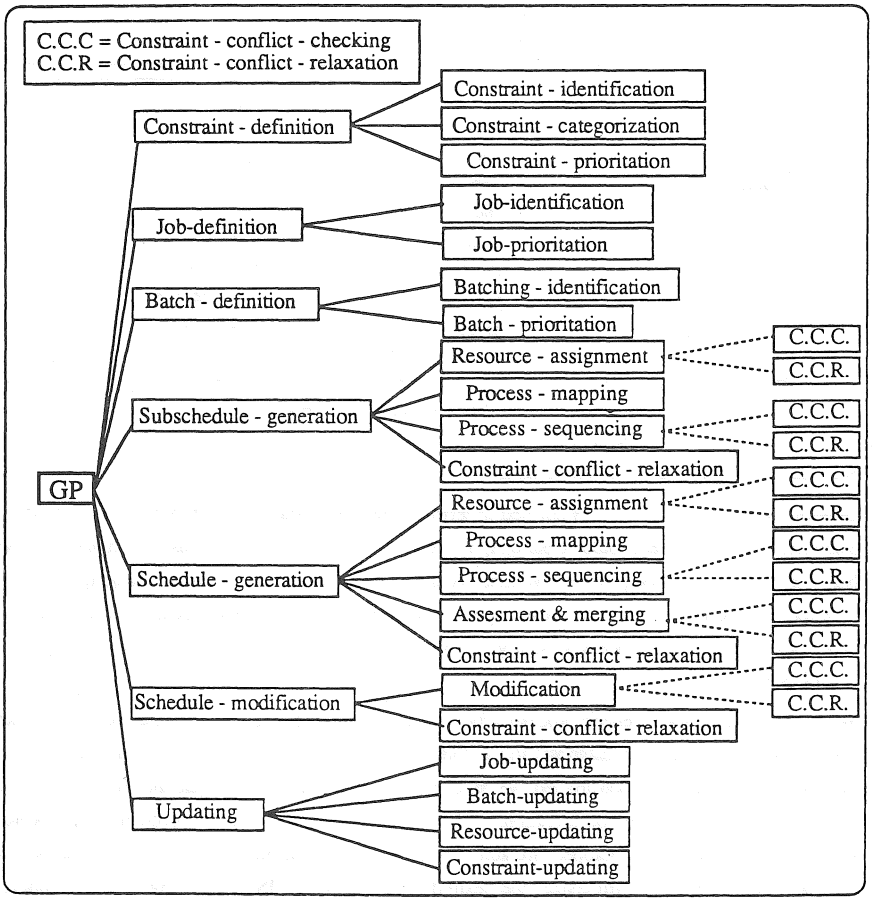


Figure 2-2. Experimental generic processes for scheduling.

2.3 Related Work

What follows is an enumeration of some related research efforts being undertaken in the area of cognitive primitives.

2.3.1 Chandrasekaran's Generic Tasks

Chandrasekaran (1986, 1987) proposes units of search control mechanisms that are very useful to construct inference engines for KBSs and to guide in the elicitation of domain knowledge. However, those units called "generic tasks" or "building blocks," though generalized for a specific type of PS tasks, are too specialized in the context of knowledge engineering. Therefore, they are very difficult to be understood by the domain expert himself.

Comparison: Our generic processes differ from Chandrasekaran's generic tasks in that our generic processes have to be combined in order to obtain knowledge-based system building blocks. In other

words, our generic processes are used to model human PS at the cognitive level and, if properly translated, can be easily understood by the domain expert. Chandrasekaran's generic tasks can be used as building blocks as they are.

2.3.2 Clancey's heuristic classification

Clancey (1985) proposes heuristic classification as a method that is useful to explain the mechanism of most KBS systems constructed in the area of analysis tasks. Again, this type of method is easily understood by knowledge engineers, but not by domain experts.

Comparison: In MULTIS we attack the problem of generalization of problem solving methods from quite a different perspective from that of heuristic classification. Heuristic classification, though very successful a method, is too general and doesn't provide enough insights for knowledge acquisition. It is difficult to believe that, because problems usually involve more than one type of task, all analysis problems can be best modeled with only heuristic classification. In MULTIS, the general assumption is made that more than just one problem solving method can be used to represent human problem solving processes in computer programs. However, if Heuristic classification is broken up in smaller component pieces, it would still be possible to provide generic processes that could be mapped to those component pieces, thus behaving as mediating representations.

2.3.3 McDermott's Half Weak Methods

Introduction of half-weak methods by professor McDermott (1988) contributed greatly to knowledge engineers in building KBSs. However, as Chandrasekaran's generic tasks, it is difficult for an average domain expert to understand his/her own PS process in terms of such half-weak methods.

Comparison: Again, these so-called half-weak methods are similar to MULTIS's building blocks, because they are helpful in modeling of inference engines for knowledge based system construction. Conceptually, half-weak methods are better employed to explain the way computers solve problems than the way humans do. However, as it is the case with heuristic classification, we believe that generic processes and generic vocabulary can be found to stand as mediating representations between the domain expert's cognitive primitives and McDermott's half-weak methods.

2.3.4 Weilinga's Knowledge Sources

Breuker and Weilinga (1989) propose to make models of human PS and call the units used in those models "knowledge sources." This is an interesting idea because those knowledge sources seem to be somewhere in between what we have called generic vocabulary and generic processes. Therefore, though KADS, the system that implements their proposal, is intended for knowledge engineers, its potential for usability by domain experts seems to be high.

Comparison: KADS's knowledge modeling framework uses generic processes and generic vocabularies (though not called that way) to model human problem solving. Although this is extremely similar to the MULTIS's modeling, KADS doesn't remember models nor it tries to adapt those models to other domain experts and was intended as a knowledge engineer's tool.

2.4 Discussion

As noticed in previous subsections, our idea of cognitive primitives depart somewhat from the context of knowledge engineering. However, we believe that cognitive primitives proposed by other

researchers are very useful as implementation tools or units for KBSs. The cognitive primitives proposed in this paper are intermediaries between human cognitive primitives and computer search control mechanism. Therefore, they provide a way to overcome the representation mismatch between domain expert and knowledge engineer, not human and machine.

2.5 An example

Let's suppose that expert A is an expert at performing a scheduling task in domain X. Let's further suppose that there is another expert B who also is an expert at performing a scheduling task, but in domain Y. Let's also assume that X is different from Y significantly. Lastly, let C be a knowledge engineer with the task of constructing two KBSs, both for modeling A's and B's problem solving knowledge in scheduling for domains X and Y respectively. Since A and B perform a common task in different domains, their task dependent primitives might also be common. Therefore, if those primitives are translated accordingly, A and B can understantd each other's problem solving processes. C, on the other hand, will have to study much deeper about each of A's and B's domains and their relation to scheduling before endeavoring to model KBSs for them respectively. In other words, C has to map his/her primitives (building blocks) to A and B and learn about X and Y in order to construct useful KBSs. Once KBSs for doing scheduling in X and Y have been modeled, their task-dependent mechanisms could be very similar, because they were modeled using C's primitives (building blocks). In this example, A and B are able to understand each other after some comparatively short discussion of X and Y. This is true because their task is the same, thus, they possess common primitives for performing it. Most of the time spent at discussing would probably be used in translating vocabulary related to X or Y. On the contrary, C has to do more effort because he has to learn new primitives (generic vocabulary and generic processes) and/or map them to his own primitives (building blocks) that are not from scheduling, but from computer programming. However, as time goes by and C constructs more scheduling KBSs, he will become more and more acquainted with such mappings and consequently more efficient at building the KBSs.

3. Building Blocks

Building blocks are transparent to the interviewee, but not to knowledge engineers who usually use them to design and construct a KBS. Though MULTIS builds a model of PS interactively with the interviewee using generic vocabulary and generic processes, a mapping of such model is made transparently to the building blocks needed to construct the inference engine of the knowledge-based system intended. This approach is consistent with the knowledge engineer's approach because they code the inference engine without showing it to the interviewee after building his/her PS model interactively.

At the end of the previous section, we compared our mediating representations with other research projects. It was concluded that the representations we introduced could still be used as mediating representations between domain expert's primitives and the primitives (building blocks) proposed by those projects. However, in this section we will introduce some experimental KBS building blocks used to prototype KBSs that perform scheduling tasks.

3.1 Summary of Mechanisms

As the name building block implies, they are mechanisms used to implement KBSs. In figure 3-1, we present some experimental building blocks. These building blocks are intended to be as general as possible in representing problem solving mechanisms identified by knowledge engineers through the years. Their combination will result in an implementable prototype for the inference engine of a KBS. Though mechanisms proposed by others (Chandrasekaran, 1986 and 1987; Clancey, 1985; McDermott, 1988) could be used as building blocks also, we have adopted this approach for simplicity reasons. Thus, we don't make any claim that the building blocks presented here cover all facets of knowledge engineering problem solving primitives. On the contrary, we propose that building blocks presented by other researchers be also used.

In figure 3-1, building blocks are divided into two types: primitive building blocks and composite building blocks. The former are more general than the later which are derived as combinations of the former. Again, the same argument introduced previously, that there exist general primitives and more specific primitives, is still valid because as can be seen, the primitive building blocks are general knowledge engineering concepts, while the combinatorial building blocks are more specific to given situations. We could also argue that there is almost no difference between the level of generic processes and this building blocks, since both endeavor to represent human cognitive primitives. However, there exist some relevant features that make distinction between them possible, that is, generic processes can be differentiated from building blocks on the following two accounts:

1) There exist no consensus in any domain or task on what generic processes could be used to successfully model problem solving for that domain or task. On the contrary, building blocks have been proposed so far that can be used to generally model knowledge engineer's problem solving.

2) Generic processes mainly model domain expert problem solving in a representation that is understandable to them, but cannot be used to explicitly implement a executable computer program. On the other hand, building blocks are representations understandable by the knowledge engineer and can be used to construct computer programs, i.e. KBSs, that implement their problem solving models.

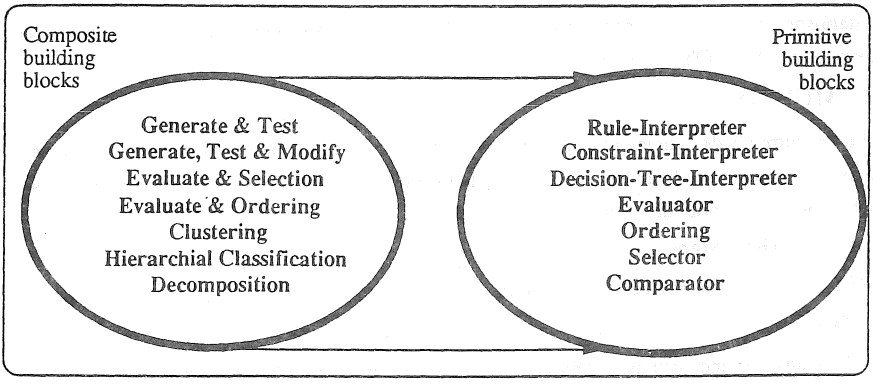


Figure 3-1. Experimental building blocks divided in composite and primitive ones.

Appendix B, gives a non-exhaustive enumeration of experimental, combinatorial building blocks being proposed in this paper.

3.2 Mapping of Generic Processes to Building Blocks

The mapping of generic processes to building blocks requires expertise of knowledge engineers in doing such mapping. Though not formally recognized previously, knowledge engineers have engaged in the task of doing such mappings when confronted with construction of KBSs. The knowledge engineer interviews domain experts, who can generally only express their problem solving in their own way, and then identifies what could be used as implementing blocks also in a rather personal manner. Consequently, a mapping is implicitly established between the domain expert's cognitive primitives and the knowledge engineers library of implementable mechanisms. The expertise used in creating this mappings can be extracted from knowledge engineers and used to perform mappings between networks of generic processes and building blocks proposed in this paper.

In appendix B, there is some guidance as to what kind of knowledge can be used for mapping generic processes to building blocks. For each generic process presented, there is an entry by the name of "mechanism identification inference," from which interview strategies could be generated to identify building blocks useful to implement the domain expert's problem solving model.

3.3 An Example

CABPRO (Schaefer et al., 1988) is a simple KBS developed to schedule processes when assembling cables. A cable could be very complex or very simple to make and very few cables could be considered standard, thus, the processes for making them vary in great degree. A schedule for making a cable is a very detailed set of instructions called a "traveler," because the person making the cable must travel from working station to station to complete the cable according to the instructions given in the traveler. Although a traveler is considered as a schedule, it must be taken into consideration that there are many travelers at any time in the factory, hence, a traveler is only a sub-schedule to the whole scheduling problem. For this reason, a traveler will be referred to as a sub-schedule.

Figure 3-2a shows what generic processes can be used to model CABPRO's task of scheduling. In figure3-2b, a translation to the CABPRO's vocabulary is given. Figure 3-2c illustrates the mapping established between the generic processes and the building blocks used to construct the inference engine of CABPRO's prototype.

4. MULTIS

We are currently developing an interview system based on the previous considerations that domain expert's primitves can be mapped automatically to knowledge engineer's primitives. What follows, what follows is a brief description of the system.

4.1 Interview Steps

MULTIS's interview strategy consists of the following four phases which are described below:

1. Task type identification. Identify the PS task the expert performs.
2. Task analysis and synthesis. Analyze the task and identify needed search control knowledge.
3. Knowledge acquisition. Elicit domain knowledge and task implementation knowledge.

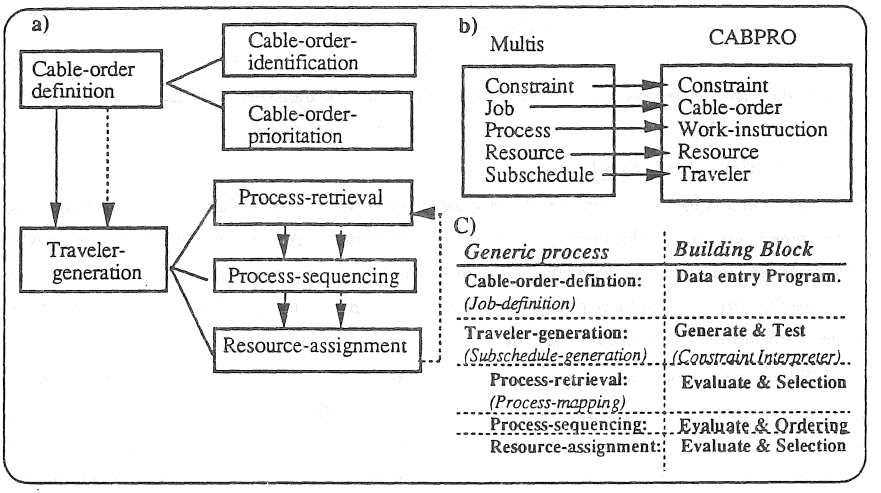


Figure 3-2. CABPRO's a) generic process network, b) generic vocabulary translation, and c) generic process to building block mapping.

4. System evaluation. Evaluate the system by imputing an already solved problem.

4.1.1 Task Type Identification

MULTIS first identifies the task the domain expert, from whom the knowledge is to be elicited, performs during PS. During this phase of the interview, the tasks for which MULTIS can elicit knowledge are currently simple planning and scheduling, but can be expanded later to cover other tasks such as diagnosis, monitoring and so on. Since a taxonomy of tasks already exists for the type of tasks being performed by humans , this phase can be implemented as a classification problem. Therefore, the interview is based on heuristics about the taxonomy of tasks. Roget (Bennett, 1985) is an earlier interview system that attempts to identify the type of task the interviewee performs before engaging in knowledge acquisition. However, it limits the type of task from which it can elicit knowledge to diagnosis by using sentence-parsing techniques that could prove somewhat inefficient if other tasks were to be added. MULTIS performs interactive interview with a user friendly interface to avoid falling into a similar problem.

Since an expert sometimes performs more than a single task in his/her field of expertise, it is necessary to identify what is the main task being performed. Once that task is identified, the rest of the tasks being performed can be ignored, because later during the interview their corresponding generic processes will be incrementally added to the model of the main task. This is a problem that hasn't been treated properly in the past because most interview systems have assumed that human experts perform only a single task. However, human PS is usually more complex, that is, it consists of a mixture of analysis and synthesis at different levels. For instance, a medical doctor mainly performs diagnosis (analysis), but he also designs treatments (synthesis) to fit special cases of patients. With this in mind, MULTIS identifies the main task the expert performs and then incrementally identifies what other tasks are being performed.

MULTIS then extracts the generic vocabulary corresponding to the task identified and translates it to the interviewee's vocabulary. As the term generic implies, a generic vocabulary is generalization of a vocabulary for a given task. Therefore, a vocabulary must be translated to reflect the interviewee's vocabulary. In MULTIS the vocabulary is translated interactively by the user himself.

4.1.2 Analysis and Synthesis

Next step is to analyze and synthesize the task identified in the previous phase. MULTIS makes use of the generic vocabulary that corresponds to the task identified in the previous step to determine what type of generic processes can be related to it. Then, with the generic processes identified heuristics are employed to single out building blocks that can be used as part of the search-control structure for the target KBS through case-based reasoning.

MULTIS makes use of case-based reasoning during its interview to reuse previous combinations of generic tasks encountered in past interviews to make search for generic tasks more efficient. The vocabulary corresponding to the expert's task and its translation are employed as indexes to search in the case base for a similar case or cases. Later, the generic task combination from the extracted case is modified with use of a graphical interface (under development) to accurately represent the task being performed by the domain expert. Finally, the generated generic task combination is analyzed heuristically to identify the right mix of building blocks that should be used as part of the search-control structure of the target KBS. The details of case-based reasoning are given in another paper (Tijerino, 1990).

PS modeling and inference engine mapping take place in this phase. Interactive modeling is implemented through the graphical interface mentioned after a prestored modifiable model has been found. The inference engine mapping is performed transparently after employing heuristics to identify building blocks useful to reflect the patterns given by the model.

4.1.3. Acquisition of Domain Knowledge

In this step, the required additional task-implementation knowledge and domain knowledge is elicited from the domain expert. For this purpose, the assumption is made that each generic process and each building block, related to the task involved, have structures that guide the processes of elicitation of domain knowledge, and task-implementation knowledge respectively. Chandrasekaran (1987) and McDermott (1988) have made similar assumptions. Put in other terms, a strategy for knowledge acquisition can be directly mapped to each generic task or each building block.

As explained before, the main characteristic of MULTIS is its ability to help acquire knowledge for various tasks. Therefore, we have concentrated mostly in the design of inference engines for those tasks through building block combination. However, acquisition of domain knowledge is also a very important issue that cannot be ignored. Consequently, we are carrying out intensive research on the mapping of existing knowledge acquisition methods to the search-control structures resulting from MULTIS's interview.

4.1.4 System Evaluation

The last phase in the interview strategy, is system evaluation. At this point, MULTIS tries to solve a problem already solved by the expert being interviewed. If the expert is satisfied with the solution then

no more questions are asked and the target KBS is constructed. Otherwise, MULTIS checks for missing or erroneous domain knowledge, or erroneous search control knowledge.

4.2 Architecture

MULTIS architecture is depicted in figure 4-1. It consists of the following modules and submodules:

1) Task Type Identification Module, which interactively identifies what is the main task type that the domain expert perform during problem solving. The purpose of this module is to help MULTIS focus on a task type to avoid unnecessary search for PS primitives from other task types

2) Generic Vocabulary Translator, which also interactively helps the domain expert translate his/her vocabulary to make the rest of the interview flow coherent with the interviewee.

3) Case-based Reasoning Module. This module uses the translation of generic vocabulary and patterns found in it to search for previous case which present some similarity with the current. Next, it interactively modifies the generic process network of the similar case with help of the Generic Process Simulator to reflect the PS structure of the interviewee. Then, it maps the generic process network obtained to implementable building blocks to yield the inference engine of the target KBS.

3.1) Generic Process Simulator. This is an user-friendly module that takes the interviewee through tutoring sessions on generic processes, shows examples of similar cases' generic process networks and simulates the flow of those networks during PS. It also provides the interviewee with powerful means to modify those networks in such a way that he/she can represent his/her own PS structure in a new generic process network.

3.2) Generic-Process-to-Building-Block Mapping Module. This module uses heuristics obtained from knowledge engineers to map generic process networks to building blocks yielding an executable inference engine.

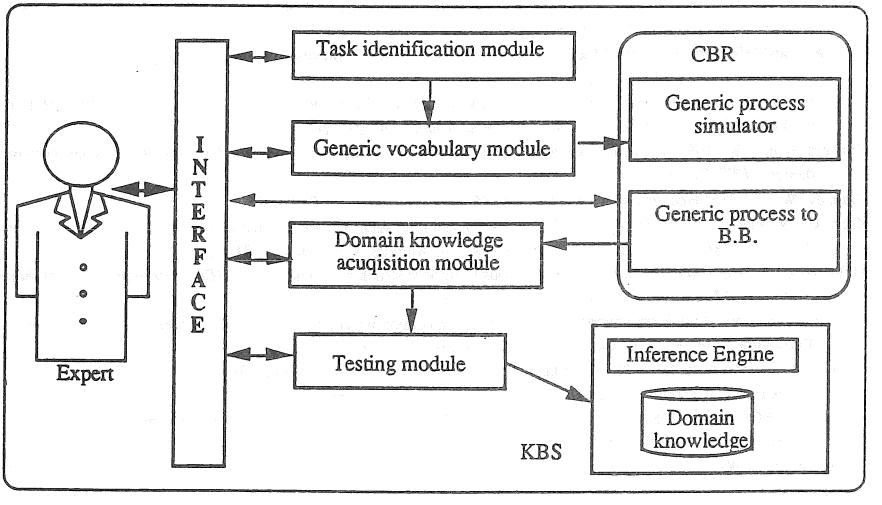


Figure 4-1. Multis Architecture.

4) Domain Knowledge Acquisition Module. This module employs interview strategies attached to building blocks and generic processes present in the current case.

5) Testing Module. This module attempts to solve an already solved case as explained in 4.1.4.

Details of each sub-module will be presented in future papers.

5. Conclusion

MULTIS is still under implementation. Once fully constructed it should be able to:

1. Engage in coherent and smooth interview with domain experts with help of generic vocabulary.
2. Model the PS processes of domain experts in various fields in the form of generic process networks.
3. Identify which prefabricated search control structures can be implemented to construct a KBS that performs the desired problem-solving.
4. Make use of strategies related to the generic processes and building blocks to elicit domain and task-implementation knowledge.
5. Help construct or quickly prototype KBSs for various tasks.
6. Make use of CBR to improve efficiency during the interview.

Currently, MULTIS possesses a case base for process scheduling with seven cases. We plan to add other cases that cover other tasks such as planning, monitoring and diagnosis. Also, mapping between generic tasks and building blocks, a very important issue, is being investigated along with the corresponding mapping to domain knowledge acquisition methods.

References

Bareiss, R. (1989). *Exemplar-Based Knowledge Acquisition.* Chandrasekaran, B. (ed) Academic Press.

Bennett J. S. (1985) ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System, *J. of Automated Reasoning* I, pp. 49-74.

Breuker, J and Wielinga, B. (1989) Models of expertise in knowledge acquisition. *In topics in Expert System Design*, G Guida and C Tasso, eds. 265-295.

Bylander, T and Chandrasekaran B. (1987) Generic tasks for knowledge-based reasoning: the "right" level of abstraction for knowledge acquisition. *Int. J. Man-Machine Studies.* 26, 231-243.

Chandrasekaran, B. (1986) Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert* 1, 23-30.

Clancey, W. J. (1985) Heuristic Classification. *Artificial Intelligence* 27, 289-350.

Kahn, G., Nowlan, S. and McDermott, J. (1985) MORE; an intelligent knowledge acquisition tool. *Proceedings of the 9th Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, 581-584.

McDermott J. (1988) Using problem-solving methods to impose structure on knowledge. *IEEE International Workshop on Artificial Intelligence for Industrial Applications.* 7-11.

McDermott, J. et al. (1990) Explorations in how to make application programming easier. *In Porceeding of 1st Japanese Knowledge Acquisition fo Knowledge-Based Systems Workshop.* 134-147.

Numao, M. and Morishita S. (1988) Scheplan—A scheduling expert for steel-making process. *In Proceedings of IEEE International Workshp on Artificial Intelligence for Industrial Applications*, 457-472.

Schaefer, R. M., Colmer, J. S. and Miley, M. (1988) CABPRO: A rule-based expert system for process planning of assembled multi-wire cables. *IEEE* CH2552-8/88 (181-186).

Rappaport, A. (1988a) Cognitive Primitives. *International Journal of Man-Machine Studies*, 29, 733-747.

Rappaport, A. (1988b) Integration of acquisition and performance systems. *Proceedings of the third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.* Banff Canada.

Tijerino, Y. A., Kitahashi, T. and Mizoguchi, R. (1990) A task-analysis interview system that uses a problem solving model. *In Porceeding of 1st Japanese Knowledge Acquisition fo Knowledge-Based Systems Workshop.* 331-344.

Appendix A: Generic Processes for Scheduling

What follows is an enumeration of the generic processes and sub-generic processes presented in figure 2-2. The order is given in the same manner as they appear in figure 2-2. Each generic process will be described according to the following description criteria:

1) a definition in terms of generic vocabulary,

2) input and output taken,

3) a general explanation of the concept they describe,

(0) *Constraint-definition*

Definition: Identify and prioritize constraints neede for scheduling.

Input: Constraints from the world.

Output: Constraints.

Explanation: With this generic task the individual performing scheduling defines what are the constraints needed in performing problem solving for scheduling.

(0.1) *Constraint-identification*

Definition: Identify constraints needed for scheduling.

Input: Constraints from the world.

Output: Constraints.

Explanation: Identify the relevant constraints that will be considered during scheduling.

(0.2) *Constraint-prioritation*

Definition: Prioritize constraint.

Input: Constraints.

Output: Contraints and priorities.

Explanation: Give an order of importance to each constraint.

(1) *Job-definition*

Definition: identify and/or assign priorities to jobs to be scheduled.

Input: Varies according to what is considered a job, but usually it is the orders taken from the customers.

Output: Jobs, Job priorities

Explanation: Job definition is a process that is performed mostly before the actual problem solving task of scheduling takes place. Although this is a beginning process, it doesn't have to be performed necessarily by the expert himself. This process deals with defining what is to be considered a job, what are the jobs to be scheduled and in some cases what are the priorities to be assigned to those jobs.

(1.1) *Job-identification*

Definition: Identify jobs to be scheduled.

Input: same as (1).

Output: Jobs.

Explanation: Job definition is a process that is performed mostly before the actual problem solving task of scheduling takes place. Although this a beginning process, it doesn't have to be performed

necessarily by the expert himself. This process deals with defining what is to be considered a job and what are the jobs to be scheduled.

(1.2) *Job-prioritation*

Definition: Assign priorities to jobs.

Input: jobs

Output: jobs, job priorities

Explanation: Prioritation of jobs can be as simple as determining the priority by a single attribute of the job or as complex as considering multiple attributes as well as other parameters external to the jobs. In essence, this process deals with assigning priorities to jobs before trying to do any scheduling.

(2) *Batch-definition*

Definition: Generate batches, with or without priorities, from jobs.

Input: Jobs, job priorities.

Output: Batches, batch priorities.

Explanation: Jobs may sometimes have similar processes that, if performed simultaneously, permit more efficient and inexpensive factory operation. In this cases, the grouping of jobs into batches is indispensable for scheduling. This process deals with generating groups or sets of jobs that can be treated as units in scheduling. A job priority is one of the many attributes that a batch might posses. However, it is a very important one when it comes to scheduling because jobs with higher priority give a starting point for scheduling. If all jobs are prioritized then there is an implicit partial order from where a schedule can be generated.

(2.1) *Batch-identification*

Definition: Generate batches from jobs.

Input: Jobs, job priorities.

Output: Batches.

Explanation: A batch is a group of jobs with one or more attributes in common. This process uses some grouping criteria to decide what jobs should be grouped. Such criteria might be the combination of various job attributes or a single one.

(2.2) *Batch-prioritation*

Definition: assign priorities to each batch.

Input: Batches.

Output: Batches, batch priorities.

Explanation: A batch priority is one of the many attributes that a batch might posses. However, it is a very important one when it comes to scheduling because batches with higher priority give a starting point for scheduling. If all batches are prioritized then there is an implicit partial order from where a schedule can be generated.

(3) *Subschedule-generation*

Definition: Generate subschedules that satisfy batch or job requirements and/or constraints.

Input: Jobs, Job priorities, Batches, Batch priorities.

Output: Subschedules.

Explanation: This process deals with generating partial, modular subschedules that satisfy local constraints to jobs or batches. Each subschedule should give a series of steps to perform the subschedule locally and in some cases identify what resources have to be used. For some scheduling problems scheduling finishes with this process, that is, by implementing a number of modular schedules problem solving is accomplished.

(3.1) *Process-mapping*

Definition: Select or generate and assign a process to a job or batch.

Input: Job, Batch, processes

Output: Processes assigned to jobs or batches.

Explanation: Process-mapping can be a very simple process, such as assigning a single process to a job or batch, or a very complex one, such as assigning numerous processes in an intricate manner. Processes can also be very simple, being composed of a small set of procedures, or very complex, a large set of procedures. For this reason process-mapping might depend on data-bases where the information about processes is prestored reflecting previous experiences gained during problem solving.

(3.2) *Process-sequencing*

Definition: Generate a sequence of processes for a job or batch.

Input: Processes, Jobs, Batches.

Output: Sequence (partial subschedule).

Explanation: Sequencing is where most of the actual scheduling takes place, because a set of ordered instructions is given in the form of a sequence. Process-sequencing deals with determining an order in which processes must be performed, where processes themselves are predetermined sets of instructions or procedures.

(3.3) *Resource-assignment*

Definition: Select and assign resources to processes, jobs, batches in sequences.

Input: Processes, Jobs, Batches, Sequences.

Output: Processes, jobs, batches in sequences with resources assigned (tentative subschedule or schedule).

Explanation: The main difference between scheduling and planning lies in that in planning the only resource considered is that of time. In scheduling the successful assignment of resources determines the overall success of the task. Resource assignment deals with choosing what resources and in what amount can be assign to a particular job, batch, processes in a sequence.

(3.4) *Constraint-conflict-checking*

Definition: Identify conflict between constraints

Input: constraints

Output: constraint conflict

Explanation: This process deals with identifying what can be treated as a conflict between the various constraints taken into consideration in problem solving, e.g. job, batch, process, resources or other global constraints. This is a process that involves some effort on the part of the problem solver to find which constraints are in conflict with others.

(3.5) *Constraint-conflict-relaxation*

Definition: Relax conflicts between constraints in within bounds assigned.

Input: Constraints in conflict, priorities.

Output: Constraints.

Explanation: When constraints are found to be in conflict with each other the problem solver must decide whether those constraints are flexible, what priorities they have and so on. Then, he/she must decide how to resolve the conflict, e.g. by satisfying only the ones with higher priority, changing the requirements in within the bounds of the constraints, etc. Constraint-conflict-relaxation is the process that is involved in making those changes so as to resolve those conflicts.

(4) *Schedule-generation*

Definition: Generate an overall schedule from jobs or batches, or merge subschedules into a single schedule.

Input: Jobs, Batches, Subschedules.

Output: Schedule.

Explanation: Schedule generation might seem too general a name for this process, however, it is intended to mean the process undertaken when the final schedule is generated from either merging subschedules or constructing one from jobs or batches. Hence this can be a process that yields the final or tentative schedule from subschedules or from either scheduling jobs or batches.

(4.1) *Assessment-and-merging*

Definition: Generate a schedule by merging subschedules.

Input: Subschedules.

Output: Schedule.

Explanation: This process deals with merging subschedules into an overall schedule of the problem being solved (the scheduled being generated). Assessment of how subschedules can be merged forms great part of this process. For most cases, modification of subschedules is performed so that they can be merged without conflicting with each other.

(4.2) *Resource-assignment*

Same as (3.3).

4.3) *Constraint-conflict-checking*

Same as (3.4)

(4.4) *Constraint-conflict-relaxation*

Same as (3.5)

(5) *Schedule-modification*

Definition: Modify a schedule into one or more schedules.

Input: Schedule, constraints, others.

Output: Schedule

Explanation: This process can be as complex as generating parallel schedules during problem solving or as simple as modifying the schedules in a small way at the end of problem solving.

(5.1) *Modification*

Definition: Modify schedule

Input: Schedule.

Output: Schedules.

Explanation: This is a more general process of modification because it serves as the modification engine itself of (5).

(6) *Updating*

Definition: Delete, modify or add jobs, batches, resources, constraints.

Input: Jobs, batches, resources, constraints.

Output: Jobs, batches, resources, constraints.

Explanation: This process deals with deleting , modifying or adding jobs, batches, resources or constraints, during or after problem solving.

(6.1) *Job-updating*

Definition: Delete, modify or add jobs.

Input: Jobs.

Output: Jobs.

Explanation: Same as (6), but only for jobs.

(6.2) *Batch-updating*

Definition: Delete, modify or add batches.

Input: Batches.

Output: Batches.

Explanation: Same as (6), but only for jobs.

(6.3) *Resource-updating*

Definition: Delete, modify or add resources.

Input: Resources.

Output: Resources.

Explanation: Same as (6), but only for resources.

(6.4) *Constraint-updating*

Definition: Delete, modify or add constraints.

Input: Constraints.

Output: Constraints.

Explanation: Same as (6), but only for constraints.

Appendix B: Experimental Building Blocks

The following is an explanation of the building blocks presented in figure 3-1.

Generate & Test.

Mechanism: Generate a schedule according to some given criteria and test the schedule for appropriateness. If the schedule is appropriate, then give it as an output to the problem. If the schedule

is not appropriate, then start from the generation process all over again. There are at least three ways of implementation of generators: Selection from a finite set, computation of formulas and production rule interpretation.

Possible primitive building blocks: Rule-Interpreter or Constraint-Interpreter.

Generate, Test & Modify.

Mechanism: Generate a schedule according to some given criteria and test the schedule for appropriateness. If the schedule is appropriate, then give it as an output to the problem. This time if the schedule is not appropriate, then modify it until it satisfies some given conditions or until it can not be modified more, in which case a new schedule is generated and the process is started all over again.

Possible primitive building blocks: Rule-Interpreter or Constraint Interpreter.

Evaluate & selection.

Mechanism: Evaluate a set of known elements according to some given criteria and select the appropriate schedule that covers prestablished conditions.

Possible primitive building blocks: Evaluator, which can be designed with other primitive building blocks such as Rule-Interpreter or Constraint-Interpreter, and Selector which can also be constructed from others.

Evaluate & Ordering.

Mechanism: Evaluate a set of elements according to some given criteria and return a set with the same elements in an order that satisfies a set of prestablished conditions.

Possible primitive building blocks: Evaluator, which can be designed with other primitive building blocks such as Rule-Interpreter or Constraint-Interpreter, and Orderer which can also be constructed from others.

Hierarchical Classification.

Mechanism: Identify to what leaf of a decision-tree an element with some given features can be classified as belonging to.

Possible primitive building blocks: Decision-Tree-Interpreter.

Decomposition.

Mechanism: Decompose a problem into smaller problems until the subproblems become such that they can be used as subschedules. Then, select the most appropriate subschedules that can be employed as the overall solution to the starting problem.

Possible primitive building blocks: Rule-Interpreter

Clustering.

Mechanism: Make groupings of elements that belong to the same category by comparing their features according to some given criteria.

Possible primitive building blocks: Comparator which can be constructed from such building blocks as Rule-Interpreter and Constraint-Interpreter.

Un ambiente para la enseñanza de la programación en lenguaje ensamblable

Enzo Chiariotti G. y David A. Koufaty A.
Departamento de Computación y
Tecnología de la Información
Universidad Simón Bolívar
Apartado 89000, Caracas 1080–A, Venezuela

Resumen

En este artículo se presentan las ventajas de simular arquitecturas para la enseñanza de la programación en lenguaje ensamblable y las características de un ambiente desarrollado para tal efecto. Este ambiente simula una arquitectura basada en el micro–procesador FUM diseñado en la Universidad Simón Bolívar. Esta arquitectura es particularmente apropiada para la docencia, ya que logra un buen compromiso entre ortogonalidad, poder expresivo y explotación del formato.

El ambiente provee además de la simulación de la arquitectura, el ensamblador de un lenguaje ensamblable definido para FUM, un enlazador y algunas herramientas para depuración.

1 Introducción

A pesar de que programar en Lenguaje Ensamblable se estila cada vez menos, es importante que el profesional de computación tenga conocimiento de los distintos niveles en la organización del computador, desde los niveles mas bajos de la lógica digital hasta los niveles superiores de los lenguajes de alto nivel.

Por otro lado, es bien conocida la relación 80–20, es decir, el 80% del tiempo de ejecución de los programas se invierte en el 20% del código de los mismos. En este marco la programación híbrida surge como una excelente alternativa para obtener una mejora sensible en la eficiencia del código generado, obteniendo una buena relación entre esfuerzo de programación y la velocidad de ejecución.

¿ Para qué tener un simulador ?

Existen variadas razones que dificultan la docencia de la programación en lenguaje ensamblable en arquitecturas reales. Aunque la base común de los lenguajes ensamblables es muy amplia, existen diferencias muy marcadas entre los diferentes lenguajes generados por las máquinas subyacentes. Estas máquinas tienden a ser complejas y presentan muchos detalles que agregan al proceso de aprendizaje un costo adicional. Por otro lado las más difundidas y económicas no son las mas simples y atractivas para la enseñanza.

Otro problema surge cuando se desea que el aprendizaje incluya una de las partes mas atractivas del mundo de la organización del computador: el manejo de excepciones; en estos casos es preferible diseñar simuladores de estos eventos, ya que de lo contrario los errores generados durante el aprendizaje pueden ser fatales para el resto de los usuarios en equipos multiusuarios y para la misma máquina. Esta perspectiva unida a la amplia difusión de los computadores personales genera una necesidad de desarollar simuladores de arquitecturas y lenguajes ensamblables.

Un simulador presenta varias ventajas respecto a las arquitecturas reales. La primera, y quizás mas importante, es su costo: si excluímos el costo de desarollo del simulador,

se pueden simular arquitecturas con el grado de complejidad deseado sobre máquinas bastantes económicas (por ejemplo computadores personales) disminuyendo el costo por equipo de laboratorio sensiblemente; incluso se extiende el alcance de los laboratorios, ya que hoy en día es común la presencia de computadores personales en el hogar, con lo cual los ejercicios pueden ser trabajados comodamente en la paz y quietud del mismo.

En segundo término, a través del simulador se puede obtener un conjunto de instrucciones sencillo y a la vez poderoso aunque no sea particularmente adecuado a las características reales de la máquina subyacente, logrando aislar los detalles innecesarios para el proceso de aprendizaje.

En tercer lugar, si el simulador es desarrollado sobre lenguajes de alto nivel, una vez programado es posible definir e implementar rápidamente diferentes máquinas que le den variadas alternativas y experiencias al estudiante. De igual forma se puede tener la misma arquitectura simulada sobre máquinas diferentes.

Por último un simulador de una arquitectura real muy costosa o de difícil acceso, trae la ventaja de poder usar el simulador para las primeras pruebas de los algoritmos y luego el recurso real para las pruebas definitivas y la versión en producción. De hecho hoy en día los diseñadores de máquinas paralelas ofrecen como primer elemento de prueba para los usuarios un simulador de la arquitectura sobre máquinas convencionales.

Como efecto de borde se produce la eliminación de los riesgos derivados de los errores en el aprendizaje, pues un error los programas de los estudiantes sólo afecta la arquitectura simulada y nunca a la máquina real, permitiendo que el producto sea robusto y resistente a las fallas de programación.

La principal desventaja de un simulador está en la velocidad de ejecución de los programas, el cual se ve seriamente disminuído. Sin embargo la velocidad de ejecución no es un factor determinante en la enseñanza.

2 El ambiente de programación FUM

FUM (*FUndamental Machine*) es un ambiente de programación diseñado en Septiembre de 1987 por los profesores Enzo Chiariotti, David Koufaty, Gustavo Lau y Alberto Torres en la Universidad Simón Bolívar. El proyecto surge como consecuencia de la crónica falta de equipo y de la flexibilidad que mostraba la idea de un simulador. Tradicionalmente el laboratorio de los cursos de Organización del Computador se desarrollaba sobre una PDP 11/45, la cual resultaba insuficiente para la demanda del curso.

En el proceso de diseño del ambiente fue necesario definir la arquitectura de la máquina, su lenguaje ensamblable y todos los elementos necesario para el desarrollo del programas (simulador de la arquitectura, ensamblador, enlazador y depurador).

2.1 La arquitectura FUM

FUM es un procesador tipo CISC (*Complex Instruction Set Computer*) inspirado fundamentalmente en dos computadoras, la PDP 11 y el Motorola MC68000. FUM se caracteriza por su ortogonalidad y la explotación del formato. Para presentar las características de la arquitectura a nivel de lenguaje de máquina convencional utilizaremos simultáneamente el lenguaje de máquina ensamblable estándar de FUM, FUMAL, descrito mas adelante.

1. Los buses de datos y de direcciones de FUM son de 16 *bits*, con lo cual el espacio direccionables es de 64K *bytes*. Sin embargo se pueden acceder operandos de 8, 16 y 32 *bytes*.

2. Posee 16 registros de 16 *bits* cada uno accesibles al usuario, de los cuales 8 son de datos (D0 – D7) y 8 de direcciones (A0 – A7). De estos últimos A6 corresponde con el apuntador a la pila o SP (*Stack Pointer*) y A7 al apuntador al programa o PC (*Program Counter*).

3. La codificación de los modos de direccionamiento se presenta siempre como un par modo-registro. FUM posee 8 modos de direccionamiento que actúan sobre uno de los grupos de registros, por lo cual la codificación de un operando requiere, excepto para algunas instrucciones, de 6 *bits*. Estos se resumen en la siguiente tabla:

Sintaxis	Modo	Registro	Nombre
Dn	0	n	Directo con registro de datos
An	1	n	Directo con registro de direcciones
@An	2	n	Indirecto con registro de direcciones
@An+	3	n	Indirecto con post-incremento
@-An	4	n	Indirecto con pre-decremento
x[An]	5	n	Indexado
@x[An]	6	n	Indexado indirecto
@@An+	7	n	Doble indirecto con post-incremento

La dirección efectiva de estos operandos se calcula fácilmente. Para los dos primeros es el registro correspondiente (Dn y An respectivamente). Para @An la dirección efectiva del operando viene dada por el contenido del registro de direcciones especificado, al igual que para el modo @An+, que además produce como efecto de borde el incremento posterior del registro. Lo mismo ocurre con @-An, excepto que el registro es decrementado antes de calcular la dirección efectiva. La dirección efectiva del modo indexado x[An] se obtiene sumando el contenido del registro de direcciones An y la constante x, mientras que en el indexado indirecto @x[An] se efectúa un indireccionamiento adicional. Por último, el modo @@An+ es similar al @An+ con un indirecionamiento adicional. La constante sumada o restada en los modos con post-incremento o pre-decremento varía de acuerdo al tamaño en *bytes* del operando, excepto para el modo 7 y para los registros A6 y A7, en cuyo caso es siempre 2.

4. FUM posee una palabra de estado de 16 *bits*, de los cuales sólo 8 son usados, con el siguiente formato:

0	0	0	0	P	P	P	0	0	0	0	T	N	Z	V	C

(a) *Bits* de prioridad PPP: representan la prioridad del procesador, que puede estar entre 0 y 7, siendo 0 la mínima. La prioridad es relevante para el esquema de entrada y salida del procesador, explicado más adelante.

(b) *Bits* de condición N, Z, V y C: reflejan el resultado de la última operación efectuada y corresponden a negativo, cero, desbordamiento (*overflow*), y acarreo (*carry*).

(c) *Trap* de *overflow* T: Habilita el *trap* automático de *overflow*.

5. El procesador identifica tres tipos de excepciones: los *traps* explícitos, los *traps* implícitos y las interrupciones. Para los primeros existen 8 instrucciones especiales, que generan el mismo número de *traps*, y el *trap* explícito de *overflow*. El *trap* de *overflow* se puede manipular además de manera implícita, habilitándolo o no por medio de intrucciones especiales. Las interrupciones son jerarquizadas por prioridades, donde cada dispositivo interrumpe enviando al procesador su prioridad, la cual es aceptada sólo si la prioridad del dispositivo es mayor que la del procesador. La prioridad del dispositivo llega al procesador por tres líneas, por lo cual la prioridad de los dispositivos está entre 1 y 7 (el 0 nunca produciría una interrupción).

   Las excepciones son atendidas empilando el PC y la palabra de estado, y cambiando el PC a la localidad indicada por los vectores de interrupción correspondiente al *trap* o interrupción. Cada vector es una localidad fija de memoria que debe contener la dirección de la rutina de servicio.

6. La entrada y salida es mapeada a memoria.

7. El conjunto de instrucciones está orientado fundamentalmente a la manipulación de operandos de 16 *bits*, aunque existen instrucciones para 8 *bits* y palabras largas de 32 *bits*. El conjunto incluye instrucciones para movimiento de datos, aritmética entera, operaciones lógicas, rotaciones y desplazamientos, control de flujo y del

CEDINFOR

micro-procesador. En el anexo se presenta una carta de ensamblaje con todo el conjunto de instrucciones. Como se observa el conjunto de instrucciones es bastante rico y flexible.

Las instrucciones requieren normalmente para su codificación de 16 *bits*, pero los modos indexado e indexado indirecto requieren de una palabra adicional. En el anexo se presenta la codificación de las instrucciones.

2.2 El micro-computador FUM-PC

El micro-computador FUM-PC implanta una computadora con una arquitectura d UNIBUS, que consta de un procesador FUM, una memoria de 32K *bytes* y 3 disposi tivos periféricos.

Los periféricos que se disponen son un teclado, un monitor y un temporizador. Ellos estan conectados al procesador a través de un PIC (*Programmable Interrupt Controller*) que envía las solicitudes de interrupción al procesador, asignando de manera estática las prioridades de los dispositivos; las solicitudes de los dispositivos con mayor prioridad son enviadas primero al procesador. Todos estos dispositivos son simulados y su efecto se presenta sobre los dispositivos reales de la máquina simuladora. Todos ellos pueden ser manipulados por medio de interrupciones y el teclado y el monitor además se pueden programar usando lazos de espera (*busy waiting*).

2.3 El lenguaje FUMAL

FUMAL (*FUndamental Machine Assembly Language*) es el lenguaje ensamblable estándar diseñado para FUM, y provee una notación simbólica conveniente para el lenguaje del procesador. Ademas de los modos nativos de la máquina se proveen cuatro modos de direccionamiento adicionales que son soportados por el ensamblador correspondiente. Estos son:

Sintaxis	Ensamblaje	Modo	Registro	Nombre
#sym	@PC+	3	7	Inmediato
sym	d(sym)[PC]	5	7	Directo relativo
@#sym	@@PC+	7	7	Directo absoluto
@sym	d(sym)[PC]	6	7	Indirecto

En los modos Inmediato y directo absoluto, la localidad siguiente a la instrucción contiene la constante sym, y d(sym) representa el desplazamiento de sym respecto a la localidad donde se colocará el mismo.

La dirección efectiva en el modo inmediato viene dado por la localidad donde se coloca la constante. En el modo directo relativo y el directo absoluto la dirección efectiva es la misma, la localidad sym; la diferencia entre ambas está en el ensamblaje y en el cálculo de la dirección efectiva a tiempo de ejecución, por lo cual el primero es adecuado para constantes relativas al comienzo del programa y el segundo para constantes absolutas. El indirecto efectúa un indireccionamiento a través de la dirección sym, de manera relativa.

2.4 Ambiente de programación

El ambiente está dirigido por menúes que facilitan la rápida integración del usuario. En el mismo se tiene acceso a la máquina virtual diseñada y a los utilitarios generales (ensamblador, enlazador, cargador y un depurador). El sistema posee dos niveles distintos: maestro y esclavo. El nivel del esclavo corresponde a la simulación de una tarjeta que posee un procesador FUM y 32 *Kbytes* de memoria RAM. Esta tarjeta es controlada por la máquina anfitriona a través de un reducido número de operaciones. El nivel de maestro corresponde a la máquina anfitriona que provee el *software* utilitario. Todos los procesos del ambiente corren en el maestro, mientras que en el esclavo sólo lo hacen los programas generados.

El ensamblador de FUMAL es un ensamblador estándar de dos pasadas, que permite entre otras cosas la resolución automática de referencias, cálculo de expresiones cons-

tantes a tiempo de ensamblaje y algunas pseudo–instrucciones para la manipulación de constantes y reservación de espacio. Además permite la referencia a símbolos definidos en otros módulos (símbolos externos) y la inclusión de archivos en el ensamblaje. El ensamblador genera dos archivos, uno con el código objeto y otro con el listado del ensamblaje del mismo y el valor de los símbolos.

El enlazador tiene interés por sí mismo, ya que con el uso de estructuras de datos adecuadas se logra realizar el enlace en una sola pasada sobre cada archivo objeto, sin necesidad de trabajo adicional. Este resuelve las referencias cruzadas y efectúa una relocalización parcial del código, generando un archivo ejecutable y un archivo con la resolución de las referencias cruzadas. La estructura del ejecutable y el objeto son similares, facilitando la creación futura de libreros.

En cuanto al resto de las herramientas, se dispone de un cargador que sólo efectúa la relocalización y copia del archivo ejecutable, y el depurador que provee facilidades para establecer múltiples puntos de parada, efectuar trazas y ejecutar paso a paso, además de permitir la modificación de registros y localidades de memoria.

El ambiente no provee un editor orientado al lenguaje pero el ambiente puede invocar un editor comercial elegido por el usuario.

Todas la características del sistema se presentan en detalle en [1, 2, 3].

3 Defectos y direcciones futuras

Desde la aparición de la primera versión en Septiembre de 1987 el ambiente ha experimentado una evolución bastante sostenida:

- 1.0 (Septiembre 1987) Primera versión.
- 2.0 (Octubre 1987) El ensamblador implementa el uso de símbolos en los modos de direccionamiento con el modo directo relativo (en la primera versión se utilizaba el

modo directo absoluto). El código objeto se representa en forma compacta (binario en lugar de ASCII) acelerando así la velocidad de carga.

- 3.0 (Noviembre 1987) Se añaden interrupciones y dispositivos periféricos. Se realizan pequeñas mejoras a la interfaz y al debugger.

- 4.0 (1988) El reconocedor de FUMAL se reescribió en Yacc [4]. Se introduce el cálculo de expresiones constantes a tiempo de ensamblaje y se modificó la notación de algunos modos de direccionamiento en FUMAL. Se aportaron mejoras significativas a la interfaz y a la documentación.

- 4.19 (Abril 1990) El sistema es sometido a una revisión completa. En el proceso de reingeniería el tamaño del código ejecutable se reduce de 102000 *bytes* a 53000 *bytes* y al mismo tiempo se aumenta la velocidad de ejecución. La simulación de los periféricos es desacoplada de la simulación del CPU. Se introdujeron en el lenguaje FUMAL las instrucciones JLO/CLO (*Jump/Call if lower*) y JHS/CHS (*Jump/Call if higher or same*), como sinónimos de JCS/CCS y JCC/CCC respectivamente. La documentación fué ampliada y actualizada.

- 4.40 (Septiembre 1990) Se introduce el enlazador y las directivas para inclusión de archivos y para la exportación e importación de símbolos.

- 4.41 (Enero 1991) Se agrega la generación de un archivo con la resolución de las referencias cruzadas durante el enlace.

El último avance significativo, el enlazador relocalizador, fué concluido en Agosto de 1990. En los últimos tres años las mejoras se han relacionado principalmente con la interfaz del ambiente y las herramientas de ayuda a la programación. Por otra parte el conjunto de instrucciones del procesador FUM se ha mantenido prácticamente inalterado desde su inicio.

El conjunto de instrucciones ha demostrado ser versátil. Sin embargo la experiencia ha puesto en evidencia la falta de algunas instrucciones. No es fácil escribir código

completamente independiente de la posición debido a la falta de una instrucción análoga intrucción a LEA del MC68000 (*Load Effective Address*): una instrucción que calcula la dirección efectiva de su operando. Las instrucciones de salto en los programas ocupan por lo general dos palabras, ya que no existen instrucciones que permitan saltos cortos relativos al PC. Las instrucciones MUL y DIV utilizan pares de registros, lo cual complica los algoritmos de asignación de registros en los compiladores.

Para el futuro ya se tiene diseñado un nuevo conjunto de instrucciones que incluye: TSB (*Test Byte*), LSW (*Load Status Word*), MEA (*Move Effective Address*), saltos cortos en una palabra (desplazamiento de 9 *bits* con signo) y cambio a la definición de MUL y DIV.

También se tiene previsto agregar un *bit* de modo protegido a la palabra de estado del procesador, lo cual va a permitir ilustrar algunos tópicos avanzados relacionados con sistemas de operación.

El ambiente de programación es susceptible de varias mejoras. Actualmente el ensamblador no permite definir macros. El depurador provee trazas, ejecución paso a paso y puntos de parada, pero no se dispone de ninguna facilidad para la depuración simbólica. Tampoco se tiene un desensamblador.

Actualmente se dispone de un prototipo de compilador de PASCAL realizado en el laboratorio de la cadena de lenguajes de programación. El uso del compilador permitir la realización de prácticas en programación híbrida. Sería también deseable disponer de compiladores de otros lenguajes de alto nivel.

El principal problema del computador FUM–PC es la falta de periféricos. Actualmente sólo se soportan en el ambiente una pantalla, un teclado y un temporizador. No existe un dispositivo de almacenamiento persistente. Se espera introducir la simulación de un disco y DMA en versiones futuras. Se está incluso considerando la posibilidad de permitir cierta configuración del computador virtual por parte del usuario, lo cual permitiría tener una familia de sistemas de cómputo basados en FUM.

La documentación del sistema es bastante completa: se dispone de manuales sobre el micro-procesador FUM, el lenguaje FUMAL, el ambiente de programación y el micro-computador FUM-PC. Sin embargo es necesario ampliarlos y documentar el código mismo del simulador.

Algunos defectos del ambiente tienen que ver con la naturaleza simulada del mismo. La simulación reduce la velocidad de ejecución considerablemente. El *software* del ambiente no corre sobre la máquina virtual misma y por razones obvias de eficiencia no sería deseable hacerlo; sin embargo la naturaleza esquizofrénica del sistema simulado puede inducir confusiones en el estudiante.

Otro problema debido a la simulación es la relación no realista entre la velocidad del procesador y la de los periféricos. Dispositivos como la pantalla son, en términos relativos, mucho más rápidos de lo que serían en la realidad. En la primera versión el procesador lograba realizar 10 accesos a la memoria (3 a 10 instrucciones) entre una interrupción de la pantalla y otra. A partir de la versión 4.19 el número ha sido aumentado sin disminuir la velocidad aparente de la pantalla. Sin embargo estas cifras distan mucho de la realidad. Mantener una relación cercana a la realidad en el simulador conllevaría una pantalla desesperadamente lenta.

Respecto a los últimos problemas mencionados no puede hacerse gran cosa por medio de *software*. Si se quieren solventar tales limitaciones hay que implementar un procesador FUM en hardware. Con el uso de una ULA comercial y la adopción de un esquema microprogramado sería posible implementar un procesador FUM en tiempo razonable y organizar un pequeño micro alrededor de él; pero esto requeriría un equipo de trabajo interdisciplinario y apoyo por parte de la institución.

4 Experiencias docentes

El desarrollo del ambiente permitió solventar cierta crisis en el laboratorio de Organización del Computador. Aumentó considerablemente la disponibilidad de tiempo de máquina del estudiante para la realización de sus proyectos. En general se percibe una mayor motivación por parte de los estudiantes hacia esta materia.

A través de la experiencia el lenguaje FUMAL a demostrado tener buenas cualidades desde el punto de vista académico. Es bastante simple sin que se pueda calificarlo como un juguete alejado de la realidad. El conjunto de instrucciones logra un buen compromiso entre ortogonalidad, aprovechamiento del formato y poder expresivo. El lenguaje FUMAL ha sido utilizado con éxito en la cadena de Lenguajes de Programación demostrando ser un ejemplo adecuado de lenguaje objetivo, tanto en la teoría como en el laboratorio del curso.

El *software* del ambiente ha servido también como material para proyectos de ingeniería de *software* de tamaño mediano, aptos para el laboratorio de la asignatura de Sistemas de Programas. Sin embargo las primeras experiencias de mejoras del ambiente con estudiantes durante el año 1989 fueron decepcionantes; no se pudo obtener un enlazador confiable y la evolución del sistema se atrasó considerablemente. Sentimos que esto se debió en gran parte a la falta de experiencia en la dirección de proyectos de este tamaño. Con la debida dirección, el enlazador fué implementado por un estudiante en una pasantía corta.

5 Conclusiones

A más de tres años de distancia del inicio de esta experiencia consideramos que el balance es francamente positivo. Los resultados obtenidos en el campo docente muestran que el esfuerzo invertido en el desarrollo del ambiente de programación FUMAL valió la pena.

Por otra parte ha demostrado su utilidad no sólo en los cursos de Organización del

Computador, ya que está siendo usado en los laboratorios de los cursos de compiladores como lenguaje objetivo y se plantea su uso en los cursos de Sistemas de Operación si se provee a FUM de modos de operación y protección de memoria.

En la actualidad se continúa el trabajo por dos vertientes. La primera consiste en evaluar las características del ambiente y del lenguaje para proveer las mejoras mencionadas. La segunda consiste en el desarrollo de simuladores de otras arquitecturas, ya que el relativo éxito de FUMAL sugiere la conveniencia de disponer de ambientes alternativos para tener distintos puntos de comparación. Actualmente se trabaja en la implementación de un ambiente similar pero basado en un procesador inspirado en la orientación RISC (*Reduced Instruction Set Computer*) y en la construcción de un simulador de una arquitectura micro-programable.

Agradecimientos

Deseamos agradecer a todos los estudiantes del curso de Organización del Computador que han acogido con gran entusiasmo el ambiente de programación FUMAL y han contribuído a la evolución del mismo aportando valiosas sugerencias. También deseamos agradecer a los estudiantes que han trabajado en la evolución del ambiente implementando nuevas características.

Un reconocimiento especial a Lisandro Ríos quien demostró una gran disposición al trabajo y logró uno de los objetivos largamente esperados.

Anexo: Carta de Ensamblaje de FUMAL

000000	NOP	000400	RTS	001000	STC
000100	EVT	000500	RTE	001100	CLC
000200	DVT	000600	HLT	00120n	STP <n>
000300	TPV	000700	RST	00130n	TRP <n>

0050dd	CLR <dst>	0054dd	TAS <dst>
0051dd	NEG <dst>	0055dd	SWP <dst>
0052dd	NOT <dst>	0056dd	SST <dst>
0053dd	TST <dst>	0057dd	ADC <dst>

0020dd	JMP <dst>	1020dd	CAL <dst>
0021dd	JEQ <dst>	1021dd	CEQ <dst>
0022dd	JGE <dst>	1022dd	CGE <dst>
0023dd	JGT <dst>	1023dd	CGT <dst>
0024dd	JCC <dst>	1024dd	CCC <dst>
0024dd	JHS <dst>	1024dd	CHS <dst>
0025dd	JVC <dst>	1025dd	CVC <dst>
0026dd	JPL <dst>	1026dd	CPL <dst>
0027dd	JHI <dst>	1027dd	CHI <dst>
0031dd	JNE <dst>	1031dd	CNE <dst>
0032dd	JLT <dst>	1032dd	CLT <dst>
0033dd	JLE <dst>	1033dd	CLE <dst>
0034dd	JCS <dst>	1034dd	CCS <dst>
0034dd	JLO <dst>	1034dd	CLO <dst>
0035dd	JVS <dst>	1035dd	CVS <dst>
0036dd	JMI <dst>	1036dd	CMI <dst>
0037dd	JLS <dst>	1037dd	CLS <dst>

004ndd	ADQ #<n>, <dst>	104ndd	SBQ #<n>, <dst>
020ndd	EQI Dn, <dst>	120ndd	XOR Dn, <dst>
021ndd	EQI #<n>, <dst>	121ndd	XOR #<n>, <dst>
022ndd	ASL Dn, <dst>	122ndd	ASR Dn, <dst>
023ndd	ASL #<n>, <dst>	123ndd	ASR #<n>, <dst>
024ndd	LSL Dn, <dst>	124ndd	LSR Dn, <dst>
025ndd	LSL #<n>, <dst>	125ndd	LSR #<n>, <dst>
026ndd	ROL Dn, <dst>	126ndd	ROR Dn, <dst>
027ndd	ROL #<n>, <dst>	127ndd	ROR #<n>, <dst>

03ssdd	AND <src>,<dst>	13ssdd	IOR <src>,<dst>
04ssdd	MOV <src>,<dst>	14ssdd	MOB <src>,<dst>
05ssdd	CMP <src>,<dst>	15ssdd	CMB <src>,<dst>
06ssdd	ADD <src>,<dst>	16ssdd	SUB <src>,<dst>
07ssdd	MUL <src>,<dst>	17ssdd	DIV <src>,<dst>

Referencias

[1] Enzo Chiariotti G. y David A. Koufaty A. El Procesador FUM. Reporte Técnico No. IT-1991-002, Departamento de Computación y Tecnología de la Información, Universidad Simón Bolívar, Caracas, Marzo 1991.

[2] Enzo Chiariotti G. y David A. Koufaty A. El Lenguaje FUMAL. Reporte Técnico No. IT-1991-003, Departamento de Computación y Tecnología de la Información, Universidad Simón Bolívar, Caracas, Marzo 1991.

[3] Enzo Chiariotti G. y David A. Koufaty A. Manual del Ambiente de Programación FUM. Reporte Técnico No. IT-1991-004, Departamento de Computación y Tecnología de la Información, Universidad Simón Bolívar, Caracas, Venezuela, Marzo 1991.

[4] Stephen C. Johnson. Yacc: Yet another compiler–compiler. Computing Science Technical Report No. 32, Bell Laboratories, Murray Hill, New Jersey, 1975.

Algoritmos Paralelos para Búsqueda de Patrones: Preproceso del patrón

David A. Koufaty A. y Alejandro Teruel L.
Departamento de Computación y
Tecnología de la Información
Universidad Simón Bolívar
Apartado 89000, Caracas 1080–A, Venezuela

Resumen

En este trabajo se presentan algoritmos paralelos para el problema de búsqueda de patrones, donde el texto y el patrón son cadenas de caracteres sobre un alfabeto finito. Estos algoritmos son en su mayoría implantables sobre arquitecturas disponibles con la tecnología actual y fueron programados en un simulador de un hipercubo, el Intel iPSC/2. Los algoritmos desarrollados paralelizan el algoritmo secuencial ingenuo y los algoritmos secuenciales optimales que construyen un índice en base al patrón. Para todos ellos se efectúa el análisis de complejidad, aceleramiento y eficiencia.

Algunos de los algoritmos paralelos son optimales si la longitud del patrón está acotada por una constante, mientras que otros, a pesar de no serlo, pueden presentar un comportamiento bastante bueno en la práctica, ya que hacen intentos por mantener el balance de carga entre los procesadores.

1 Introducción

El problema de búsqueda de patrones se puede plantear de la siguiente forma: dada una cadena de caracteres de longitud n denominada texto, y otra de longitud m denominada patrón, con $m < n$, encontrar las ocurrencias del patrón en el texto. En nuestro caso nos interesa sólo el problema cuando el patrón y el texto son cadenas de caracteres sobre un alfabeto finito Σ y deseamos encontrar todas las ocurrencias del patrón, utilizando para ello arquitecturas paralelas con memoria distribuída.

Existen muchos algoritmos optimales tanto secuenciales como paralelos. En cuanto a los algoritmos secuenciales, el más ingenuo requiere un orden $O(n*m)$ en el peor caso y los optimales

requieren un orden de $O(n+m)$, basando su eficiencia en la construcción de índices auxiliares para la búsqueda, y pueden ser divididos en dos grupos: aquellos que construyen los índices en base al patrón y aquellos que lo hacen en base al texto. La mayoría de los algoritmos paralelos optimales no se basan en los algoritmos secuenciales optimales, sino que resuelven el problema de una manera completamente diferente.

Uno de ellos [MS84] resuelve en forma paralela el problema de hallar la primera ocurrencia del patrón en el texto y se basa en el algoritmo secuencial de Boyer y Moore [BM77]. El algoritmo efectúa una división del texto similar a la presentada en este trabajo, utilizando secciones de tamaño constante sobre una máquina que posee 8 procesadores y memoria compartida. El algoritmo propuesto es incompleto pues no muestra como resolver el problema de una ocurrencia en la frontera de dos secciones. Tampoco se efectúa un análisis del comportamiento teórico del algoritmo y de la magnitud de la pérdida de linealidad a medida que se aumentan los procesadores ni de la conveniencia o no de seleccionar secciones de tamaño fijo.

Otros algoritmos presentados son sólo de interés teórico, pues el modelo computacional utiliza memoria compartida con lecturas concurrentes y, en algunos casos, acceso concurrente para la escritura de una misma constante; ademas, el mínimo número de procesadores es $n/\log^2 n$. Algunos de ellos no son optimales, entre los cuales está el de Karp y Rabin [KR87], mientras que los algoritmos de Galil [Gal84, Gal85], Vishkin [Vis85] y Berkman et al. [BBG+89] si lo son, obteniendo en este útlimo un orden de $O(\log\log m)$ con $n/\log\log m$ procesadores usando el modelo computacional descrito.

En nuestro caso presentaremos algoritmos que pueden ser implementados en las arquitecturas disponibles en la actualidad. El modelo computacional usado será una máquina tipo MIMD (Multiple Instruction Multiple Data) con memoria distribuída, en la cual se puede solapar la comunicación con el cómputo. En algunos caso se presenta el algoritmos para un módelo SIMD (Single Instruction Multiple Data).

Para efectuar el análisis de los algoritmos utilizaremos varias métricas, a saber, el tiempo de ejecución $T(n)$, el aceleramiento (*speedup*) S, el número de procesadores P, el costo $C(n)$ y la eficiencia E del algoritmo.

Para el problema de búsqueda de patrones con un texto de longitud n (variable) y un patrón fijo de longitud m denotaremos con $T_{opt}(n)$ el tiempo tomado por los algoritmos secuenciales optimales y con $T_{par}(n)$ el tiempo tomado por un algoritmo paralelo usando P procesadores. Para la comunicación denotaremos con M_p al número de mensajes enviados por cada procesador más los que recibe del anfitrión y con M_c el número de caracteres transmitidos en estos mensajes.

Los algoritmos presentados aquí han sido programados en su totalidad en el lenguaje C [KR78] en un simulador de un hipercubo de Intel, el iPSC/2 [Int]. La programación y detalles

de los mismos se presenta en [Kou91].

En las secciones 2 y 3 presentaremos las versiones paralelas del algoritmo ingenuo y los algoritmos que preprocesan el patrón, respectivamente. Luego en la sección 4 se efectúa el análisis de aceleramiento y eficiencia de estos algoritmos. Por último se presentan las conclusiones.

2 Algoritmos ingenuos

El algoritmo secuencial más ingenuo para resolver el problema consiste en intentar encontrar el patrón en cada una de las posiciones del texto. Para ello se efectúa una comparación entre el patrón y m caracteres del texto, comenzando en cada una de las posiciones del texto. De esta manera el algoritmo sería:

Para $i = 1$ hasta $n - m + 1$ hacer
  Si pat[1]...pat[m] es igual a text[i]...text[$i + m - 1$]
  entonces reportar una ocurrencia en la posición i

Varios algoritmos paralelos ingenuos se obtienen a partir de éste. Si disponemos de $n-m+1$ procesadores, podemos obtener uno paralelizando el lazo exterior, donde el procesador i es responsable de buscar el patrón a partir de la posición i del texto. En el mejor de los casos si disponemos de memoria compartida con lecturas simultáneas, el algoritmo es de orden $O(m)$ y costo $(n - m + 1) * m$.

Un algoritmo de orden constante se puede obtener paralelizando ambos ciclos, si disponemos de $n * m$ procesadores con una memoria compartida en la cual se pueden efectuar escrituras simultáneas de una misma constante 0. Para ello se requiere de un arreglo de tamaño n con cada entrada inicializada en el valor 1. El procesador (i,j), donde i varía desde 1 hasta n y j de 1 a m, es responsable de comparar text[i] con pat[j] (aunque para ser precisos, i no debe ser mayor que $n - m + 1$). En caso de ser diferentes coloca en 0 la entrada i del arreglo. Las entradas que permanecan en 1, indican las posiciones de ocurrencia del patrón. Como se ve, este algoritmo es de $O(1)$ y tiene un costo igual al caso anterior.

En los dos casos anteriores se trata de paralelizar el acceso al texto. Desde otro punto de vista podríamos tratar de paralelizar el acceso al patrón. Aunque el resultado obtenido obliga a descartar la solución, el enfoque no deja de tener interés. La idea en este algoritmo consiste en tener m procesadores, responsable cada uno de la posición del patrón correspondiente. Cada procesador debe comparar el caracter del patrón asociado a su posición con todos los caracteres del texto y determinar las posiciones en donde coinciden. Luego se debe proceder a determinar si existen m posiciones encontradas por los m procesadores que sean consecutivas (es decir el

primer procesador indica i, el segundo $i+1$, hasta el último que indica $i-m+1$). Para resolver este problema el algoritmo propuesto se basa en un *pipeline* de procesadores, donde existe además un anfitrión (*host*) que tiene comunicación con el primero y el último de los procesadores, de manera que el anfitrión y los procesadores conforman un anillo.

Los caracteres del texto entran al *pipeline* y cada nodo, dependiendo de su estado, compara el caracter del patrón que posee con el caracter del texto que le llega. Luego los caracteres son enviados al próximo procesador. Para determinar las ocurrencias los caracteres viajan junto con una etiqueta, que asegura que al salir del *pipeline* la etiqueta tiene valor m si y sólo si hay una ocurrencia del patrón que finaliza en ese caracter. El estado según el cual los procesadores deciden comparar o no los caracteres que le llegan depende del caracter anterior que le llegó al procesador. El tiempo de ejecución del algoritmo es de orden $O(n+m)$, que viene dado por el tiempo durante el cual se llena el *pipeline* mas el tiempo que tardan en salir por él n elementos. Su costo es $nm+m^2$.

A pesar de que el aceleramiento de los dos primeros algoritmos es bastante bueno, todos los algoritmos son de costo mayor o igual a $O(n*m)$. Los primeros dos algoritmos son optimales respecto al algoritmo secuencial ingenuo, pero no así con respecto a los algoritmos secuenciales optimales que, como indicamos al principio, son de orden $O(n+m)$. Con esto obtenemos que la eficiencia de estos algoritmos es:

$$E = \frac{n+m}{n*m} = \frac{1}{n} + \frac{1}{m}$$

lo que indica que mientras más procesadores se usen (o de mayor longitud sea el patrón o el texto) el algoritmo hace peor uso de los recursos.

3 Algoritmos que preprocesan el patrón

En esta sección se mostrarán algoritmos paralelos que utilizan parcialmente alguno de los algoritmos secuenciales que preprocesan el patrón. El uso de estos algoritmos secuenciales es local a cada procesador y en muchos casos es irrelevante cual de ellos es.

Existen tres algoritmos secuenciales que preprocesan el texto y que dan origen a muchas variantes y combinaciones, estos son el algoritmo de Knuth, Morris y Pratt [KMP77], el de Boyer y Moore [BM77] y el de Karp y Rabin [KR87]. Los tres algoritmos son optimales en tiempo con $T_{opt} = O(n+m)$, aunque los dos últimos lo son en el caso promedio, pues en el peor caso son de orden cuadrático $O(n*m)$ como el ingenuo. Recientemente Sunday [Sun90] presenta una familia de algoritmos que recoge las ideas de los dos primeros y genera varios algoritmos de búsqueda, resultando casos particulares los algoritmos clásicos de Knuth, Morris y Pratt y el de

Boyer y Moore.

En el algoritmo de Knuth, Morris y Pratt el texto es examinado de izquierda a derecha. El algoritmo tiene dos pasos: el primero de ellos consiste en procesar el patrón para computar una tabla que luego ayudará en el segundo paso, que es la búsqueda propiamente dicha. La fase de búsqueda del algoritmo se inicia alineando el patrón con el comienzo del texto. Las comparaciones van de izquierda a derecha. Mientras sean exitosas, se avanza una posición en el texto y en el patrón, hasta llegar al final del patrón cuando encontramos una ocurrencia. Si ocurre una falla, se desplaza el patrón a la derecha un número de posiciones, indicado por la tabla precomputada, que asegure que no se pierde ninguna ocurrencia; esto se hace sin desplazar el apuntador al texto. La cantidad por la cual se desplaza el patrón en el caso de una falla corresponde a la máxima ocurrencia del patrón en sí mismo, cantidad que garantiza que se puede continuar la búsqueda desde esa posición sin comparar los caracteres anteriores con el texto.

El algoritmo para búsqueda de patrones que más versiones ha generado es quizás el de Boyer y Moore. A diferencia del algoritmo de Knuth et al, este algoritmo al alinear el patrón en alguna posición del texto examina sus caracteres de derecha a izquierda, comenzando en el último caracter del patrón. A medida que los caracteres alineados coinciden se examinan los caracteres a la izquierda, hasta llegar al principio del patrón cuando se ha encontrado una ocurrencia. Si existe una diferencia se desplaza el patrón a la derecha de acuerdo a una cantidad precomputada y las comparaciones se reinician al final del patrón. La cantidad por la cual se desplaza el patrón viene dado por la cantidad máxima entre dos heurísticas: la de ocurrencia y la de coincidencia. La primera indica para cada caracter del alfabeto cual es su posición más a la izquierda en el patrón. La segunda es similar a la heurística del algoritmo de Knuth et al., pero considerando que los caracteres son examinados de derecha a izquierda.

El algoritmo de Karp y Rabin es un algoritmo probabilístico, que usa la idea de clasificación (*hashing*) para la búsqueda de patrones. El algoritmo básicamente computa una función (firma) para el patrón y lo compara con el valor de la función en cada cadena de m caracteres del texto. Inicialmente se computa la firma del patrón y de los primeros m caracteres del texto. Dada la firma de una posición $i-1$, el cálculo de la firma de la posición i se puede hacer en orden constante. Sin embargo si dos firmas son iguales (hay una colisión) no se garantiza que haya una ocurrencia, por lo cual es necesario una comparación adicional con el texto.

La idea principal de los algoritmos paralelos de esta sección consiste en paralelizar el segundo paso de los algoritmos secuenciales optimales. Estos algoritmos consisten todos de un preproceso y luego una búsqueda. En base a esto, el problema de hallar las coincidencias entre un texto de longitud n y un patrón de longitud m, se puede dividir en varios subproblemas, cada uno con

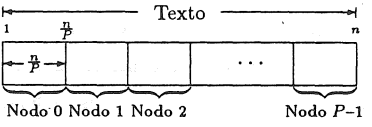


Figura 1: Partición del problema

el mismo patrón pero con una sección de texto de menor tamaño y, como veremos adelante, un trabajo posterior adicional.

El algoritmo genérico propuesto en base a esta observación es:

1. El anfitrión asigna a cada procesador secciones de texto.
2. Cada procesador aplica localmente uno de los algoritmos optimales de de búsqueda secuencial en cada sección asignada.
3. Cada ocurrencia es reportada al anfitrión.

Los procesadores son responsables de encontrar las ocurrencias dentro de las secciones de texto a él asignadas. Un ejemplo de tal asignación se muestra en la figura 1, donde a cada procesador se le asigna una sección de n/P caracteres. Es evidente que el número de procesadores P debe ser a lo sumo n, caso en el cual a cada procesador le corresponde un caracter del texto.

En este algoritmo surgen dos problemas básicos:

1. Cómo se efectúa la asignación de texto a los procesadores, y
2. Cómo se detectan las ocurrencias del patrón que estén en el límite entre dos secciones consecutivas asignadas a dos procesadores distintos.

La asignación de texto a los procesadores puede ser prefijada (estática) o dinámica. La diferencia entre ellas está íntimamente ligada al problema de balanceo de carga: cómo evitar que un procesador no tenga asignado un trabajo mientras los demás sí.

El segundo problema puede ocurrir cuando los últimos k caracteres de una sección coinciden con los primeros k caracteres del patrón y los primeros $m-k$ caracteres de la siguiente sección coinciden con los últimos $m-k$ caracteres del patrón. De esta forma la coincidencia no detectada por ninguno de los dos procesadores asignados a estas dos secciones, tal como lo muestra la figura 2. Es evidente que las únicas ocurrencias que un procesador no puede detectar son aquellas que comienzan en una posición mayor de $n/P - m$ relativa al comienzo de la sección.

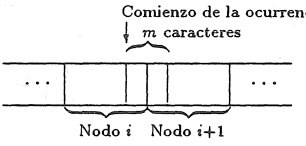


Figura 2: Ocurrencia no detectada en el algoritmo genérico

3.1 Asignación de secciones

Si asignamos estáticamente las secciones al momento de comenzar la ejecución el tamaño total de las secciones que le corresponderán a cada procesador. Si disponemos de P procesadores el número total de caracteres que deberá analizar cada uno es de n/P. En este caso lo más conveniente es que todos los caracteres asignados a un procesador sean consecutivos, generando una única sección de tamaño n/P para cada uno, evitando la proliferación de ocurrencias en la frontera. Si denotamos con q el tiempo necesario para resolver el problema de las ocurrencias en la frontera de una sección, el orden de algoritmo un algoritmo con este esquema es:

$$T_{par}(n) = T_{opt}\left(\frac{n}{P}\right) + q = O\left(\frac{n}{P} + q + m\right)$$

La asignación estática sufre de un problema básico en todo algoritmo paralelo: balanceo de carga. El trabajo de cada procesador es fijado a priori, por lo cual es posible que un procesador quede ocioso mientras otros trabajan, ya que, por ejemplo, en el algoritmo de Knuth et al., la constante del número de comparaciones es 2 en el peor caso, pero en el mejor caso es exactamente 1. Esto indica que puede ocurrir que algunos procesadores terminen su trabajo en la mitad del tiempo que otros.

Para tratar de aminorar el efecto que tiene la diferencia en la dificultad de cada sección, la siguiente alternativa es no asignar las secciones a priori, sino entregar a cada procesador secciones de tamaño constante s a medida que finalice el análisis de la sección que posee. s debe ser menor que n/P para que el algoritmo mejore con respecto al anterior. El número total de secciones de tamaño s a enviar es n/s. Supondremos en las siguientes subsecciones que el número de secciones de texto asignadas a cada procesador es en promedio $n/(sP)$. Observese que en este caso el orden de la búsqueda en cada sección es $O(s)$, a lo cual hay que agregar el proproceso del patrón una sola vez. En este caso el orden de un algoritmo es:

$$T_{par}(n) = (T_{opt}(s) + q)\frac{n}{sP} + m = (O(s) + q)\frac{n}{sP} + m = O\left(\frac{n}{P} + q\frac{n}{sP} + m\right)$$

Este esquema de asignación aventaja al estático:

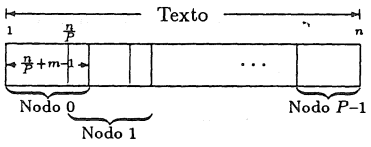


Figura 3: Solapamiento de secciones

1. El balanceo de carga es más justo.
2. El espacio necesario en cada procesador es $O(s + m)$, el texto y el patrón, mientras que en los algoritmos anteriores es $O(n/P + m)$. Si s es mucho menor que n/P, el ahorro es significativo.

3.2 Ocurrencias en la frontera

La primera alternativa consiste en asignar los caracteres en la frontera por duplicado, solapando las secciones asignadas. Bastaría con que el solapamiento sea de $m - 1$ caracteres, ya que de ser menor aún habría ocurrencias no detectadas y de ser mayor se reportarían dos veces. Así, a cada sección de texto asignada se le deben agregar los $m - 1$ caracteres correspondientes al solapamiento. Por ejemplo si utilizamos un esquema de asignación estática combinado con el solapamiento de las secciones obtenemos la asignación presentada en la figura 3. El orden adicional de detectar las ocurrencias en la frontera es $O(m)$, ya que el texto es ahora $m - 1$ caracteres mayor.

Estudiando detenidamente el algoritmo de Knuth, Morris y Pratt, se pueden hacer las siguientes afirmaciones:

1. El algoritmo puede examinar más de una vez cada caracter del texto (comparándolo con el patrón).
2. Una vez que avanza al próximo caracter del texto no retrocede. Esto indica que el apuntador al texto siempre se incrementa.
3. Cuando el algoritmo llega al final del texto, se tiene un índice sobre el patrón que indica la máxima coincidencia entre el texto y el patrón al final del texto. Esto corresponde con la semántica del índice del patrón, ya que mantiene en todo momento la máxima coincidencia hasta la posición actual en el texto.

Si anexamos otros caracteres al final del texto la búsqueda se podría reiniciar con sólo saber el valor final del índice j del patrón con el cual concluyó la búsqueda sobre el texto inicial. La búsqueda a partir de la posición $j + 1$ del patrón y el primero de los nuevos caracteres.

La otra alternativa consiste en explotar esa particularidad para evitar las secciones solapadas. Cada procesador recibe su sección de texto de acuerdo al esquema de asignación utilizado. y aplica el algoritmo de Knuth et al. Al finalizar cada procesador sabe cual fue la coincidencia máxima entre el patrón y la parte final de su sección y la envía al procesador que posee la sección siguiente para que continúe haciendo la búsqueda. Este último paso consiste en aplicar el mismo algoritmo pero partiendo con un valor inicial para el índice del patrón, siguiendo el algoritmo hasta la posición $m-1$ del texto, pues de ahí en adelante ya el procesador efectuó la búsqueda (en realidad basta con buscar hasta que el índice del texto supere al del patrón, pues en ese caso la ocurrencia finalizaría al menos en la posición m y éstas ya han sido reportadas). El orden adicional para las ocurrencias en la fronteras es $O(m)$, ya que se aplica el algoritmo de Knuth et al. con un texto y un patrón de longitud m.

A pesar de no mejorar el orden de la búsqueda de las ocurrencias en la frontera, este esquema presenta varias ventajas con respecto al de solapamiento de secciones:

1. En el caso de modelos con memoria distribuída el algoritmo es mejor, pues se evita la transmisión adicional de $(m-1)(P-1)$ caracteres y tener por duplicado algunas secciones del texto.
2. De trabajar en un modelo con memoria compartida no se necesitaría el acceso concurrente a ninguna sección del texto.

Como se desprende del esquema, la topología ideal para la misma es una topología lineal, pues la comunicación sólo es necesaria entre procesadores que tienen secciones consecutivas.

Para garantizar que cada procesador pueda finalizar la búsqueda iniciada por el anterior es conveniente que la sección asignada sea de al menos de tamaño $m-1$, ya que de lo contrario sería necesario la coloboración de más de un procesador para detectar las ocurrencias compartidas, lo cual aumenta la complejidad comunicacional. Por ello es recomenedable que $m \leq n/P$, lo cual nos da una indicación sobre el número de procesadores del algoritmo:

$$P \leq n/m$$

Como se ve, la diferencia en tiempo entre los algoritmos esta en determinación del esquema de asignación de secciones, mientras que la diferencia en comunicación depende del de detección de ocurrencias en la frontera. Combinando los equemas de asignación estática y dinámica con los de solapamiento y colaboración se obtienen varios algoritmos, cuyo comportamiento teórico se resume en las tablas 1 y 2. En el caso del número de caracteres no se incluyen los $n/P+m$ caracteres indispensables que debe recibir cada procesador.

	Asignación estática	Asignación dinámica
Tiempo	$\frac{n}{P} + m$	$\frac{n}{P} + m\frac{n}{sP}$
Eficiencia	$\frac{n+m}{n+mP}$	$\frac{n+m}{n+mn/s}$

Tabla 1: Tiempo y Eficiencia

	Asignación estática		Asignación dinámica	
	Solapadas	Colaboración	Solapadas	Colaboración
Mensajes M_p	1	2	$2\frac{n}{sP} - 1$	$3\frac{n}{sP}$
Caracteres M_c	$m - 1$	1	$m\frac{n}{sP} - 1$	$4\frac{n}{sP} - 1$

Tabla 2: Comunicación

3.3 Descentralización de la administración

La idea de este algoritmo es descentralizar la administración del texto, manteniendo en lo posible el balance de carga entre los procesadores. Uno de los problemas que surge en el algoritmo general propuesto antes, consiste en el posible cuello de botella en el que se puede convertir el anfitrión, quien en ese caso se desenvuelve como administrador del texto. Una solución al problema consiste en combinar la asignación estática y la dinámica. El algoritmo descrito a continuación es sobre una topología de hipercubo y tiene sentido cuando el número de procesadores disponibles es al menos 8 (cubo de dimensión 3), ya que las ideas expuestas requieren que cada nodo posea al menos 3 vecinos.

La idea principal es asignar una sección de tamaño $k * n/P$ caracteres a un grupo de k procesadores, al cual denominaremos macro-procesador, y que entre todos hallen las ocurrencias en esa sección. El caso descrito a continuación consta de 4 procesadores por macro-procesador.

Este grupo debe tener la particularidad que uno de los procesadores es vecino del resto de los elementos del grupo. Al procesador central del macro-procesador lo denominaremos M-nodo. Este será el administrador de la sección de texto asignada al macro-procesador, convirtiéndose así en un anfitrión local para los otros procesadores de macro-procesador.

Cada uno de estos M-nodos recibe los caracteres correspondientes a todo el macro-procesador, es decir, $4(n/P)$ caracteres. Las acciones que toma este nodo son las mismas que tomaba el anfitrión en el caso del algoritmo de asignación dinámica, excepto que sólo sobre los nodos del macro-procesador. Además de administrar, cada M-nodo también efectúa las funciones de

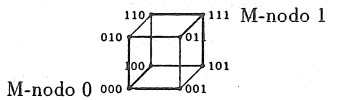


Figura 4: Cobertura de un cubo de dimensión 3 por macro-procesadores

búsqueda, ya que con tres procesadores dependientes es poco el trabajo de administración.

Para efectuar la implementación del algoritmo en el iPSC/2 se requiere primero encajar la topología del algoritmo en el hipercubo. Se escogen como M-nodos todos aquellos procesadores cuya representación binaria termine con los dígitos 000 ó 111. Los tres vecinos de cada M-nodo en el macro-procesador se obtienen invirtiendo cada uno de los tres últimos bits de la representación binaria de su número en el hipercubo. Para el caso de un M-nodo de la forma X...X000 sus vecinos son el X...X001, el X...X010 y el X...X100. De esta forma queda cubierto un cubo de dimensión 3 por cada dos macro-procesadores, como lo muestra la figura 4.

En el caso de los caracteres en la frontera entre dos M-nodos, se puede seguir cualquiera de las políticas descritas antes: secciones solapadas o colaboración entre procesadores. Si deseamos usar el segundo método, debemos hacer algunas consideraciones sobre la asignación de segmentos a los M-nodos, de manera de minimizar la distancia entre ellos. Es posible lograr que todas las secciones consecutivas estén en procesadores a distancia 1, excepto dos que está a distancia 3.

La idea para minimizar la distancia entre los M-nodos consiste en encajar dos anillos, un anillo en cada uno de los dos sub-hipercubos de dimensión $d-3$ que forman los macro-procesadores que terminan en 000 y 111, y luego establecer la conexión entre ellos. Para encajar los anillos en cada uno de los sub-hipercubos usamos el código de Gray y escribimos una lista con los procesadores adayacentes consecutivamente. Las secciones de texto son asignadas a los M-nodos de la primera lista desde el primero al último y luego a los de la segunda lista desde el último hasta el primero. De esta manera todos los M-nodos con secciones consecutivas están a distancia 1, excepto los últimos de cada lista que están a distancia 3, ya que los últimos procesadores de ambas listas son iguales a excepción de los tres últimos bits, que son 000 para uno y 111 para el otro.

Para el caso de los caracteres en la frontera de las secciones dentro de un macro-procesador, dado que los procesadores están muy cercanos entre sí, resulta muy conveniente la utilización del algoritmo de colaboración entre procesadores y no el de solapamiento.

La complejidad computacional de este algoritmo permanece invariable respecto a los algoritmos de secciones dinámicas, ya que se efectúan exactamente el mismo algoritmo de distribución de secciones de tamaño s, excepto que localmente.

La complejidad comunicacional en cuanto al número total de caracteres enviados M_c, aumenta con repecto a los algoritmos de secciones dinámicas, ya que, como se observa en el algoritmo, el texto se retransmite en la red; una vez lo envía el anfitrión a los M-nodos y luego los M-nodos lo retransmiten al resto de los procesadores. Por esto se crea un factor adicional de n en el total de caracteres enviados. El número de comunicaciones M_p efectuadas por cada procesador se mantiene como en el algoritmo de secciones dinámicas en los procesadores no M-nodos y aumenta en los M-nodos, ya que deben efectuar el trabajo de distribución al resto. Sin embargo el número total de transmisiones en la red disminuye con respecto al de secciones dinámicas, ya que, las transmisiones de distribución están contadas dos veces (una en los M-nodos y otras en los no M-nodos); si descontamos de las transmisiones de los M-nodos las necesarias para la distribución de las secciones a los procesadores no M-nodos, los M-nodos sólo efectúan una comunicación con el anfitrión. La diferencia está en que el M-nodo recibe todas sus secciones en una sola comunicación (aunque en un principio no sabe cuales son).

Es importante destacar que todas las transmisiones en el algoritmo, a diferencia de los anteriores, se pueden realizar en paralelo y por canales de comunicación distintos, ya que se aseguró por la topología usada que la distancia entre los procesadores que cooperan es mínima y lo hacen por canales distintos.

3.4 Balanceo automático de carga

En este algoritmo la idea básica es hacer una asignación inicial de texto estática, pero que a medida que los procesadores terminen, le ofrezcan ayuda a los procesadores que aún están ocupados. Los procesadores ocupados deben decidir si aceptan la ayuda o no. La ayuda es rechazada si el tiempo que le tomaría al nodo procesar la fracción que le quede es inferior al tiempo que se tardarán los caracteres enviados en llegar al otro procesador mas el tiempo en que éstos serán procesados. Si denotamos con k el número de caracteres por procesar y con j el número de caracteres a enviar, el óptimo de tal relación se obtiene con:

$$T_{opt}(k-j) = T_{transmisión}(j) + T_{opt}(j)$$

Este tiempo de transmisión lo podemos expresar como $t_0 + t * j$, donde t_0 es el tiempo inicial necesario para establecer la comunicación (*setup*) y t es el tiempo requerido para la transmisión de cada caracter (ancho de banda del canal). Además podemos expresar el tiempo optimal de procesar i caracteres como $t_p * i$, donde t_p es el tiempo promedio de los algoritmos secuenciales para examinar cada caracter. Entonces se requiere que:

$$(k-j) * t_p = t_0 + j * t + j * t_p k = \frac{t_0}{t_p} + \left(\frac{t}{t_p} + 2\right) * j \qquad (1)$$

Como era de esperar el resultado es muy dependiente de la velocidad relativa de transmisión con repecto al cómputo. Por otro lado, sabemos que j debe ser mayor o igual a m, pues por las mismas razones expuestas antes es conveniente que ninguna sección sea menor de m caracteres, con lo cual el mínimo número de caracteres con lo cual sería compartida la carga es:

$$k \geq \frac{t_0}{t_p} + \left(\frac{t}{t_p} + 2\right) * m \qquad (2)$$

En base a esto se compartiría la carga si k es al menos el valor indicado en la ecuación 2 y los caracteres a enviar serían, en el caso óptimo, los indicados por el valor de j en 1. Observemos que si, por ejemplo, el valor de t_0 es 1000 veces mayor que t_p, el valor de t es 50 veces peor que el de t_p y estamos buscando un patrón de longitud 30, el mínimo número de carateres con los cuales se comparte la carga es 2560, de lo cual parece razonable esperar que el algoritmo sólo tiene utilidad con textos muy grandes.

Una consideración necesaria sobre el algoritmo es cómo se ofrece o rechaza la ayuda entre los procesadores. Una posibilidad sería que los procesadores libres efectuaran una difusión de su disponibilidad y los procesadores ocupados que así lo deseen soliciten la ayuda; para el caso que más de un procesador solicite la ayuda, se debe esperar la confirmación o rechazo del procesador que la ofrece. Otra posibilidad que involucra menos comunicación en la red, consiste en mantener en el anfitrión el estado de trabajo de cada procesador. Cuando un procesador queda libre envía la notificación al anfitrión y éste le indica algún procesador ocupado. Si el procesador indicado por el anfitrión no acepta la ayuda se marca como desocupado y se repite el proceso hasta que todos los procesadores queden desocupados.

Como se ve la implementación eficiente de este algoritmo requiere conocer el tiempo de comunicación promedio entre los procesadores, dato que no se obtuvo, por lo cual este algoritmo no fue implementado.

La complejidad computacional de este algoritmo es difícil de determinar de manera exacta, ya que depende de las características del patrón y el texto. Evidentemente resulta inferior a los de secciones dinámicas. Si los procesadores ocupados deciden no distribuir la carga entre los libres, la complejidad se mantiene igual a la de los algoritmos de secciones estáticas; pero en el caso de haber distribución de carga, se espera que sea ligeramente superior. Lo mismo ocurre con la complejidad comunicacional. Por ello no podemos dar una fórmula exacta para la complejidad promedio de este algoritmo, pero será un valor intermedio entre las complejidades de los algoritmos de secciones estáticas y los de secciones dinámicas, mas cercano a los primeros que a los segundos.

4 Análisis de los algoritmos

4.1 Aceleramiento y eficiencia

Como hemos calculado previamente tenemos básicamente dos ordenes distintos en los algoritmos, los presentados en la tabla 1.

4.1.1 Secciones estáticas

Para el caso de los algoritmos de secciones estáticas el aceleramiento es:

$$S = \frac{n+m}{n/P+m}$$

Nos interesa observar el comportamiento de esta función. Para ello veremos tres casos de patrones: patrones de longitud 1, patrones de longitud n/P y patrones de longitud n.

1. Para el caso de patrones muy pequeños tomaremos como caso extremo $m = 1$. Desarrollando para el caso de que el número de procesadores está entre 2 y n, obtenemos que el aceleramiento está entre $S = 2(n+1)/(n+2) \cong 2 = P$ y $S = (n+1)/2 = (P+1)/2 \cong P/2$. Esto nos indica que la eficiencia para el caso de patrones pequeños está entre $1/2$ y 1.

2. En el otro caso extremo tenemos que $m = n$, con lo cual $S = \frac{n+n}{n/P+n} = \frac{2P}{P+1} \cong 2$ En este caso el aceleramiento es siempre cercano a 2, por lo cual el algoritmo está muy lejos de hacer un buen uso de los recursos que posee y a medida que se aumentan los procesadores la eficiencia disminuye.

3. En el caso medio tomamos $m = n/P$ (o equivalentemente $P = n/m$). Obtenemos $S = \frac{n+n/P}{2n/P} = \frac{P+1}{2}$. Esto indica que el algoritmo obtiene una eficiencia superior a $1/2$ si logramos mantener el número de procesadores inferior al número de secciones de tamaño m que podemos formar.

En conclusión podemos afirmar que los algoritmos de secciones estáticas presentados tienen una eficiencia teórica superior a $1/2$ si el número de procesadores usados en el algoritmo es $P \leq n/m$. El algoritmo se comporta ineficientemente en cualquier otro caso, llegando a obtener en el peor caso un aceleramiento de apenas 2, aún con un gran número de procesadores, por lo cual es recomendable disminuir el número de procesadores a medida que la longitud del patrón se acerca a la del texto.

4.1.2 Secciones dinámicas

El aceleramiento obtenido para los algoritmos de secciones dinámicas es:

$$S = \frac{n+m}{n+mn/s}P$$

Presentaremos el análisis en base al tamaño de la sección s.

1. En el caso de secciones de tamaño reducido, $s = 1$, el aceleramiento es:

$$S = \frac{(n+m)P}{(m+1)n} < \frac{(n+m+1)P}{(m+1)n} = \left(\frac{1}{m+1} + \frac{1}{n}\right)P$$

   lo cual es bastante deficiente, ya que el aceleramiento y la eficiencia resultan inversamente proporcional a las longitudes del texto y el patrón.

2. En el caso de tomar secciones de m caracteres obtenemos que el aceleramiento es $S = \frac{n+m}{n+n}P = \frac{n+m}{2n}P$. Sabemos que el patrón tiene longitud entre 1 y n, con lo cual la eficiencia esta entre $1/2$ y 1.

3. En el caso de secciones de n/P caracteres, obtenemos el algoritmo de secciones estáticas.

Resulta evidente que el comportamiento de los algoritmos mejora sensiblemente con la selección de secciones de texto relativamente grandes, siendo recomendable tomar un mínimo de m caracteres por sección. Este es un resultado esperado ya disminuyendo el tamaño de las secciones, el número de secciones de texto resulta mayor y con ello la cantidad de caracteres en las fronteras, los cuales son revisados dos veces por el algoritmo.

Algo que se deduce de ambos esquemas de asignación de secciones es que los algoritmos están muy cerca de los optimales si los patrones son pequeños relativos al texto. En los casos prácticos de búsqueda son frecuentes los patrones de pocos caracteres (a lo sumo algunas decenas de ellos) y texto de millones de caracteres.

4.2 Comportamiento límite

La ley de Amdahl coloca un límite al aceleramiento que puede ser obtenido en los algoritmos paralelos. Si definimos α, denominado fracción de Amdahl, como la fracción del algoritmo que debe correr secuencialmente (no es paralelizable), tenemos que:

$$T_{par}(n) = \alpha T_{opt}(n) + \frac{(1-\alpha)T_{opt}(n)}{P}$$

y

$$S = \frac{P}{1+(P-1)\alpha}$$

Por lo tanto hay un límite en el aceleramiento alcanzado dado por:

$$\lim_{P\to\infty} S = \frac{1}{\alpha}$$

Realmente α no es una constante, sino que depende del tamaño del problema. A continuación efectuaremos el estudio del comportamiento de los algoritmos cuando tenemos patrones muy grandes.

4.2.1 Secciones estáticas

En el caso de los algoritmos de secciones estáticas, la fracción de Amdahl corresponde a dos factores. Si suponemos que el texto está inicialmente distribuído uniformemente entre los procesadores, a excepción de la diferencia de 1 caracter que se puede dar si n no es divisible por P, la contribución del algoritmo a la fracción de Amdahl está determinado por el tiempo de distribución del patrón y el trabajo adicional de las ocurrencias en la frontera (que en cualquiera de las dos soluciones presentadas es $O(m)$). En el hipercubo la distancia que tienen que viajar los mensajes está acotada por $\log_2 P$, por lo que el tiempo de transmisión del patrón es $O(m \log_2 P)$. En este caso la fracción quedaría determinada por

$$\alpha = \frac{O(m\log_2 P) + O(m)}{T_{par}(n)} = \frac{O(m\log_2 P) + O(m)}{O(n/P + m)} = O\left(\frac{m(\log_2 P + 1)}{n/P + m}\right)$$

Si consideramos que m está acotada por una constante, lo cual en la práctica es cierto, y que disponemos de un número finito de procesadores tenemos que

$$\lim_{n\to\infty} \alpha = 0$$

por lo cual se puede concluir que el algoritmo obtiene un aceleramiento cercano al lineal, $S = P$, para texto de longitud muy grande si la memoria de los procesadores (para un número fijo de procesadores) es suficientemente grande para almacenar la fracción de texto que le corresponde.

Una conclusión similar se obtiene analizando el comportamiento de la función del aceleramiento derivada antes, ya que con las mismas consideraciones anteriores:

$$\lim_{n\to\infty} S = \lim_{n\to\infty} \frac{n+m}{n/P+m} = P$$

4.2.2 Secciones dinámicas

En los algoritmos de secciones dinámicas se tiene que:

$$S = \frac{(n+m)sP}{(s+m)n}$$

En este caso el comportamiento límite viene dado por:

$$\lim_{n\to\infty} S = \lim_{n\to\infty} \frac{(n+m)sP}{(s+m)n} = \frac{s}{s+m}P$$

El valor $s/(s+m)$ es siempre menor que 1 y se acerca a 1 a medida que la sección de texto s es mayor, con lo cual el aceleramiento es menor que P y la eficiencia menor que 1. Para mejorar la eficiencia las secciones de texto deben ser relativamente grandes. Cuando la sección es n/P se obtiene el mejor aceleramiento, por lo cual el algoritmo de secciones estáticas es mejor en el caso límite.

4.3 Complejidad comunicacional

Como la comunicación entre procesadores y con el anfitrión es de orden lineal en el tamaño del problema, la complejidad computacional del algoritmo es al menos lineal (caso óptimo). Sin embargo existen diferencias en cuanto al número de comunicaciones necesarias en cada algoritmo que alteran el comportamiento real de los mismos. Todos los algoritmos requieren como mínimo la transmisión del texto y el patrón; como se requiere que todos los procesadores posean el patrón, el mínimo número de caracteres transmitidos a cada procesador es $n/P + m$ con una transmisión.

Observando la tabla 2, podemos afirmar que los algoritmos de solapamiento de secciones resultan más económicos en el número de transmisiones efectuadas, ya que al haber colaboración, el número de transmisiones en cada procesador aumenta por la cantidad de secciones que recibe el procesador, lo cual es 1 en el caso de secciones estáticas y $n/(sP)$ para el de secciones dinámicas. Sin embargo resultan más costosos en cuanto a número total de caracteres enviados, ya que el número de caracteres enviados, adicionales al texto y el patrón, en los algoritmos de secciones solapadas es cerca de m veces el número de secciones asignadas al procesador, mientras que en los de colaboración es una constante por el número de secciones asignadas. Esto es sencillo de explicar, lo primero se debe a que en los algoritmos con solapamiento de secciones los procesadores no se comunican entre sí para hallar la solución y lo segundo a que en los algoritmos de colaboración no se duplica ninguna sección del texto.

Sin embargo los algoritmos de secciones estáticas son en todos los aspectos de comunicación mejores que los de secciones dinámicas. Es de esperar que el comportamiento del algoritmo de descentralización sea similar al de secciones dinámicas.

Un hecho interesante se obtiene si tabla anterior denotaremos con N_p el número de secciones que recibe cada procesador en los algoritmos (1 para el de secciones estáticas y $n/(sP)$ para el de secciones dinámicas). La misma tabla, pero expresada de esta forma nos queda:

	Secciones estáticas		Secciones dinámicas	
	Solapadas	Colaboración	Solapadas	Colaboración
Mensajes M_p	N_p	$2N_p$	$2N_p - 1$	$3N_p$
Caracteres M_c	$(m-1)N_p$	N_p	$mN_p - 1$	$4N_p - 1$

En la tabla se observa que en todos los algoritmos el número de comunicaciones es directamente proporcional al número de secciones que recibe cada procesador. Además en los algoritmos de secciones solapadas, se depende adicionalmente del tamaño del patrón.

5 Conclusiones

Algunos de los algoritmos presentados en este trabajo no son implantables con la tecnología disponible. Tal es el caso de los dos primeros algoritmos ingenuos de las sección 2, los cuales usan $O(n)$ y $O(n * m)$ procesadores respectivamente y necesitan memoria compartida para lectura. El tercer algoritmo ingenuo obtiene resultados muy pobres que obligan a descartarlo. El resto de los algoritmos presentan ventajas y desventajas relativas.

Si denotamos con N_s el número de secciones en las cuales se divide el texto en cada algoritmo (P o n/s) y con N_p el número de secciones que recibe cada procesador (1 o $n/(sP)$) podemos escribir el tiempo y la eficiencia de la tabla 1 como:

$$T_{par}(n) = \frac{n}{P} + m * N_p$$

$$E = \frac{n+m}{n+m*N_s}$$

En base a esto, es evidente que la complejidad de los algoritmos aumenta proporcionalmente al número de secciones asignadas a cada procesador. Por ello el algoritmo de secciones estáticas resulta mejor que el de secciones dinámicas, teóricamente. Sin embargo, el algoritmo de secciones dinámicas puede obtener mejores resultados en la práctica para algunos casos, ya que en problemas que presenten mal comportamiento con respecto al balanceo de carga, utilizando secciones suficientemente grandes (por ejemplo, $n/(2P)$) es posible superar a los algoritmos de secciones estáticas. Lo mismo ocurre con los algoritmos de descentralización y de balanceo automático de carga.

La selección entre los algoritmos de secciones solapadas y los de colaboración entre vecinos dependerá fundamentalmente del tamaño del patrón, ya que con patrones muy cortos el solapamiento genera duplicación de pocos caracteres, mientras que con patrones largos los caracteres a transmitir pueden ser demasiados. Si ademas el texto puede ser modificado y queremos evitar la posible inconsistencia entre los caracteres distribuídos resulta mejor utilizar los algoritmos de

colaboración y evitar la duplicación de partes del texto. Es necesario considerar además el costo real de las comunicaciones adicionales a causa de la colaboración y determinar si estas resultan mas costosas que el tiempo de transmisión de los caracteres solapados.

En algoritmo de descentralización puede ser útil en el caso de que el texto esté distribuído en varios discos fijos, de manera que pueda ser accesado concurrentemente por un grupo de procesadores (tal es el caso del hipercubo). En este caso es necesario transmitir al resto de los procesadores las secciones de texto que analizarán y resulta conveniente el esquema de asignación de secciones.

El rebalanceo automático puede ser superior a todos los algoritmos anteriores, pero requiere un esfuerzo de entonación sobre las constantes que representan el tiempo promedio de analizar un caracter y el tiempo promedio real de transmisión del mismo.

El Intel iPSC/2, arquitectura usada para la prueba de los algoritmos, presentó la ventaja de la gran independencia existente entre la arquitectura y los algoritmos: los mensajes son enviados y recibidos en diversas modalidades que permiten un manejo transparente de la comunicación e independiente de la arquitectura. Por otra parte al perder el control del enrutamiento de mensajes no existe forma de indicar por cuál canal debe salir un mensaje. Para asegurar un manejo eficiente de la topología de hipercubo es necesario diseñar algoritmos muy particulares (tal es el caso del algoritmo de descentralización) que aseguren un buen uso de la misma; con lo cual los algoritmos dejan de ser "portables" de una máquina a otra, ya que la topología usada en los mismos en muy rígida. Lamentablemente no se contó con otros simuladores u otras máquinas con las cuales efectuar una comparación de sus ventajas y desventajas durante la programación.

Además de los algoritmos presentados en [Kou91] se presentan algoritmos paralelos basados en la versión secuencial que construye los índices en base al texto.

Referencias

[BBG+89] Omer Berkman, Dany Breslayer, Zvi Galil, Baruch Schieber, y Uzi Vishkin. Highly parallelizable problems. In *Proc. of the* 21^{th} *ACM Symposium on Theory of Computing*, páginas 309–319, 1989.

[BM77] Robert S. Boyer y J. Strother Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[Gal84] Zvi Galil. Optimal parallel algorithms for string matching. In *Proc. of the* 16^{th} *ACM Symposium on Theory of Computing*, páginas 240–248, 1984.

[Gal85] Zvi Galil. Optimal parallel algorithms for string matching. *Information and Control*, 67:144–157, 1985.

[Int] Intel Corp. *iPSC/2 Simulator Manual.*

[KMP77] Donald E. Knuth, James H. Morris, y Vaugham R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[Kou91] David A. Koufaty A. Algoritmos paralelos para búsqueda de patrones. Tesis de Maestría, Universidad Simón Bolívar, Caracas, 1991.

[KR78] Brian W. Kernighan y Dennis M. Ritchie. *The C Programming Language.* Prentice–Hall, Inc., Englewood Cliffs, New Jersey, 1978.

[KR87] Richard M. Karp y Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–261, 1987.

[MS84] Peter Moller-Nielsen y Jorgen Staunstrup. Experiments with a fast string searching algorithm. *Information Processing Letters*, 18:129–135, 1984.

[Sun90] Daniel M. Sunday. A very fast substring search algorithm. *Communications of the ACM*, 33(8):132–142, 1990.

[Vis85] Uzi Vishkin. Optimal parallel pattern matching in strings. *Information and Control*, 67:91–113, 1985.

A Note on the Definition of Visibility for Protected Class Members in C++

Alberto Nava B., Enzo Chiariotti
Departamento de Computación y
Tecnología de la Información
Universidad Simón Bolívar
Apartado 80659, Caracas 1070-A, Venezuela
emsca!usb!beto@sun.com
emsca!usb!enzo@sun.com

Resumen

En este artículo se muestra como la definición de la visibililidad de los miembros protegidos en el lenguaje de programación C++ causa ciertos problemas en el modelaje de sistemas. Se presentan algunos ejemplos que abstraen situaciones de la "vida real" en los cuales se puede observar como la ocurrencia de dichos problemas afecta la reusabilidad del software. Finalmente se proponen dos definiciones alternas que no sufren de tales inconvenientes y que permitirían mejorar este lenguaje para la programación orientada por objetos.

Abstract

This article shows some problems with the definition of visibility of protected members in the C++ programming language. We present some examples which abstract "real life" situations in which these problems affect code reusability. Finally, we propose two alternative definitions of visibility rules for protected members that not suffer of such problems.

Introduction

Protected fields in C++ [1] are a useful mechanism for partial data hiding while keeping efficiency in implementations. Briefly, if class B is derived from class A then protected members of class A are visible to class B objects. A class designer can define some members as protected, granting access to derived class implementors while at the same time hiding those members from end users.

Typical candidates to be protected are those members which have to deal with object data structure. We want to hide them from end users since we don't want user code to depend on the data structure of the class implementation. However, we also want to provide efficient access to data structures to the implementors of derived

classes. A further use of protected members is to provide some operations to the designers of derived classes that should be hidden from the end user. We could even define a class in which all constructors are protected, so that users cannot declare objects of the class but can use its derived classes.

As Snyder says [3], a class interface can be considered as an agreement with other users: public members are for everyone, protected members are only for implementors of new derived classes and private members should be only for the designer of the original class.

```
class A {
public:
...        // members for all end users
protected:
...        // members for derived class implementors
private:
...        // private stuff you should never know
}
```

Note that a change in the protected interface will affect derived classes but not end users. This is one of the reasons for which Snyder state that a class interface is like an agreement. "Protected" is an agreement between class designer and derived class designers.

Problems with protected fields in C++

Despite the above, the usefulness of protected fields in C++ is to some extent compromised by the following characteristic of the definition:

"A friend or a member function of a derived class can access any public or protected static member of a publicly derived class. A friend or a member function of a derived class can access a protected non-static member of a base class only through a pointer to, reference to, and object of the derived class of which it is friend or member (and pointers to, and objects of classes publicly derived from this class)" [2]. As a result of this, non-static protected members of a parameter of the same class as the base class are not accesible by a friend or a member function of the derived class.

Suppose we want to model bird relationships and we know that *Emperor Penguins* don't like animals either too different from them or from other habitat. Moreover, *Emperor Penguins* don't like *King Penguins* :

```
class Bird {
public:
     virtual int Likes(Bird& other);
     virtual int CanFly() { return 1; }
     void  Show();
protected:
     String Habitat;
     String WhoAmI;
     Photo  Picture;
};


class Emperor_Penguin : public Bird {
public:
     Emperor_Penguin() {
        Habitat = "Antartic"; WhoAmI = "Emperor_Penguin";
        Picture.Load("Emperor_Penguin.map");
     }
     int CanFly()  { return 0; }
     int Likes(Bird& other) {
          return other.Habitat == Habitat &&
                 Picture.Diff(other.Picture) < TOO_DIFFERENT &&
                 !(other.WhoAmI == "King_Penguin")
     }
}
```

Appealing as it may seem, the above code will not compile: function `Likes()` of class `Emperor_Penguin` receives a reference to a `Bird` as a parameter; the body of the function makes use of the protected members `Habitat`, `Picture` and `WhoAmI` which is not allowed by the AT&T definition of C++ [2,73].

We can patch the problem in the above example by making `Emperor_Penguin` a friend class of `Bird`.

```
class Bird {
     friend class Emperor_Penguin;
     friend class King_Penguin;
public:
     virtual int Likes(Bird& other);
     virtual int CanFly() { return 1; }
     virtual void Show();
private:
     String Habitat;
     String WhoAmI;
     Photo  Picture;
};


class Emperor_Penguin : public Bird {
...       // same as above
};
```

However, as Stroustrup says [1,9], use of friends should be avoided whenever possible. Moreover, each time you add a new class derived from `Bird` you have to make that class a friend class of `Bird`.

Another possibility is to define, for each protected member, a public function which returns a reference to it:

```
class Bird {
public:
     virtual int Likes(Bird& other);
     virtual int CanFly() { return 1; }
     virtual void Show();
     String& Habitat() {return _Habitat;}  // Maybe const String&
     String& WhoAmI()  {return _WhoAmI;}
     Photo&  Picture() {return _Picture;}
private:
     String _Habitat;
     String _WhoAmI;
     Photo  _Picture;
};
```

```
class Emperor_Penguin : public Bird {
public:
     Emperor_Penguin() {
        Habitat = "Antartic"; WhoAmI = "Emperor_Penguin";
        Picture.Load("Emperor_Penguin.map");
     }
     int CanFly()  { return 0; }
     int Likes(Bird& other) {
          return other.Habitat() == Habitat &&
                 Picture.Diff(other.Picture()) < TOO_DIFFERENT &&
                 !(other.WhoAmI() == "King_Emperor")
     }
}
```

However, this solution presents some problems:

- When defining these functions in the public interface, their names, arguments and return value types are now part of the agreement between the implementor and the end user. As an example, the return type of `Picture()` , the type `Photo`, is now part of the agreement with the end user. A change in the type used to represent a bird picture (the type of `_Picture`) can affect user code that uses the return value of `Picture()` for some purpose.

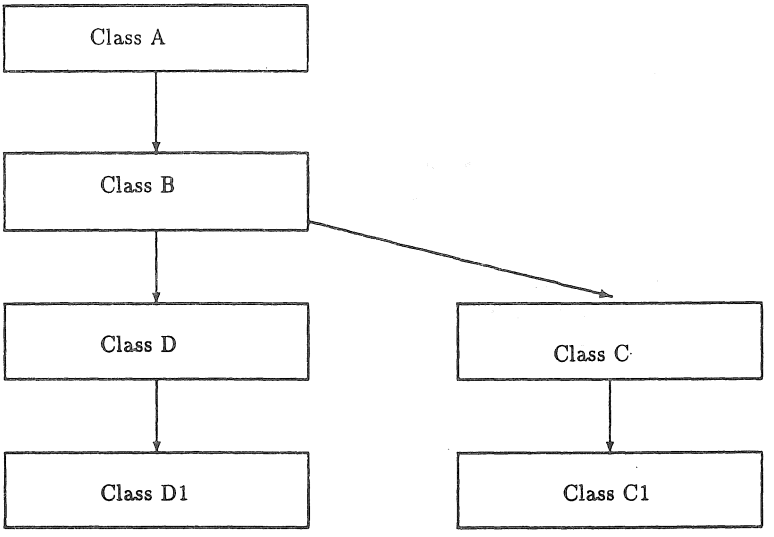
- The introduction of the `WhoAmI()` operation in the public interface goes against the object oriented style of programming, since it encourages users to write code which uses "type-field" solutions [1,9][1,199].

Given the unsatisfactory nature of the solutions proposed, we suggest that a different policy for protected member visibility be defined, to make the first version above correct.

Two Alternative Definitions

A possible alternative definition could be: "A friend or member function of a derived class D can access any public or protected member of a base class B through a pointer to, reference to, or object of class B (and pointers to, references to, or objects of classes publicly derived from B)".

Although this definition is very simple, some objections could be raised regarding its permissiveness. As an example, consider the visibility of protected members from the point of view of friends and member functions of class D in the following hierarchy tree:



According to the AT&T definition a member or friend of class D can access protected non-static members of A and B only through instances of D and D1. On the other hand, our previous definition allows access to protected members of A and B through instances of any class in the tree hierarchy. To date we are not sure whether permit access to protected members of A and B through instances of C and C1 is a good idea or not.

Another possible definition would be to allow access to those members only through instances of classes in the path of direct inheritance; in our example this would imply that class D members could access protected members of A and B through instances of A, B, D and D1, but not of C and C1.

Conclusions

C++ tries to achieve a good balance between many conflicting requirements; not an easy task. However we think that the definition of visibility of protected members in the AT&T definition is not in harmony with other aspects of the language and that some practical examples are affected by this. We suggest that the prohibition on the use of protected members of other instances of the base class must be lifted. This seems to be more consistent with other characteristics of C++.

References

[1] Bjarne Stroustrup. *The C++ Programming Language*, Addison-Wesley, 1986.

[2] Unix System V AT&T C++ Language System, Release 2.0.

[3] Alan Snyder. "Encapsulation and Inheritance in Object-Oriented Programming Languages". OOPSLA '86 Proceedings, September 1986.

Hipermedio para apoyar Explicaciones de Sistemas Expertos

Andrés Lepe A. y Pedro Hepp K.

Departamento Ciencia de la Computación, Escuela de Ingeniería, Pontificia Universidad Católica de Chile

Vicuña Mackenna 4860 Casilla 6177 (147), Santiago - Chile.
FAX 56025524054
E-mail: phepp@dcc.puc.cl

Resumen

Tradicionalmente se han entendido las explicaciones de sistemas expertos como mecanismos estrechamente relacionados con el motor de inferencias. En este artículo se propone separar el conocimiento del razonamiento e incorporar conocimiento de apoyo a las explicaciones y justificación del comportamiento del sistema, tendiente a elevar el nivel de abstracción. La utilización de herramientas de hipermedios en este sentido permite enfatizar el carácter pedagógico de un sistema experto, a la vez que refuerza la credibilidad del sistema como resolutor de problemas. Las ideas expuestas han sido probadas en sistemas expertos de diagnóstico.

1. Introducción

Idealmente, un sistema experto (SE) debiera poder explicar su razonamiento en los términos requeridos por el usuario, así como los expertos humanos lo hacen. Esta es una tarea bastante difícil. En primer lugar, el humano, al explicar el razonamiento, no sólo puede describir la solución que propone, la secuencia de deducciones hechas y la estrategia global, sino que también puede describir los fundamentos teórico-prácticos que justifican sus deducciones, pero que no se utilizan en el razonamiento en sí. Un SE, por su parte, podría explicar la secuencia de deducciones a partir de la base de conocimiento (BC), y de hecho, este es el enfoque más utilizado en los sistemas actuales, pero difícilmente podría explicar los fundamentos teóricos de su BC[1,3,13,16].

En segundo lugar, el experto humano puede explicar el razonamiento en términos requeridos por el usuario, porque es capaz de ponerse en el lugar de su interlocutor (empatía), y determinar su nivel de conocimiento para generar una explicación apropiada.

Otro aspecto muy importante es determinar el *objetivo* que se quiere lograr con la explicación: ésta no será la misma si se quiere *convencer* a un usuario escéptico de la solución o recomendación propuesta por el experto, que si se quiere *enseñar* el proceso de razonamiento a un usuario receptivo, o simplemente, si se quiere *clarificar* conceptos[16].

Finalmente, el experto humano puede recurrir a ejemplos, papel y lápiz, o medios más sofisticados (fotografías, planos, gráficos, etc.) para apoyar y transmitir su explicación (por ejemplo un médico puede utilizar una radiografía para explicar el diagnóstico o el tratamiento que su paciente debe seguir).

Para los SE`s, generar explicaciones en términos del usuario involucra un aspecto **cognoscitivo**, ya que es necesario manipular un modelo del usuario[1,3,7,13] para evaluar y determinar el nivel de conocimiento requerido, y un aspecto de **interfaz humano-computador**[1,3], ya que se requiere el manejo de representación gráfica de la información (gráficos, animaciones, video, etc.) como elemento de apoyo, del mismo modo como el humano se vale de lápiz y papel.

En resumen, el poder explicar el conocimiento en un sistema experto implica responder a las siguientes preguntas: *qué explicar* (acerca del razonamiento: decisiones del SE; acerca del conocimiento del dominio: conceptos), *cómo generar la explicación* (a partir de la BC y conocimiento del dominio adecuadamente representado) y *cómo transmitirla* (determinar nivel de abstracción y canal de comunicación).

2. Trabajo Precedente

El tema de los mecanismos de explicación es relativamente nuevo en el campo de la Inteligencia Artificial y se ha orientado más bien a explicar las decisiones del SE, solucionar el aspecto de *cómo generar la explicación* (relacionado con el mecanismo de inferencia), y la transmisión a través de lenguaje natural[11,15,16].

El enfoque más generalizado en los mecanismos de explicación se ha centrado en explicar la inferencia en torno a la BC del sistema, desplegando el camino recorrido o por recorrer para llegar hasta la solución, lo cual ha probado ser inefectivo en los sistemas basados en reglas: estos mecanismos pueden servir para describir el comportamiento del programa, pero no puede decirse que lo justifiquen[3, 6, 12,16] ya que aunque la justificación de una BC es parte del conocimiento usado para desarrollar el programa, este conocimiento no queda explícitamente representado. La justificación se entiende como un subtipo de explicación cuyo objetivo no sólo es hacer el proceso más claro, sino específicamente apoyar la decisión del SE, demostrar que es correcta y, por lo tanto, convencer al usuario[13].

En este contexto, el sistema MYCIN[1,3,8,16] provee un modo de preguntar por qué (WHY) ciertas decisiones fueron hechas o cómo (HOW) ciertas sentencias serán alcanzadas. MYCIN expresa sus explicaciones enteramente en términos de reglas y objetivos, apoyados con explicaciones textuales estáticas asociadas a cada regla[8,16].

Los mecanimos WHY y HOW han sido generalizados como mecanismos de explicación en los sistemas actuales, sobre todo en las herramientas de producción de sistemas expertos o shells, difiriendo sólo en aspectos gráficos y de contenido de la información desplegada, entre uno y otro sistema. Por ejemplo, Nexpert, KEE, Exsys, Level5, CL/1.

Por su parte, en XPLAIN se adopta un enfoque de programación automática[1,3,6,16]. La idea es relacionar los procesos de diseñar el sistema y obtener explicaciones coherentes de él. Se especifica el *modelo del dominio* (que corresponde muy estrechamente a lo que Clancey llama conocimiento-estructural), los datos y los *principios del dominio*, estrategias de control y manipulación (el conocimiento-estratégico de Clancey) para generar el programa experto. La integración de los aspectos descriptivos y prescriptivos se usa para generar explicaciones del comportamiento del sistema.

Hay dos ideas fundamentales nacidas del grupo de MYCIN que se concretaron en NEOMYCIN de Clancey[8,10,16] y del sistema XPLAIN, de Swartout.

La primera es que proveer explicaciones en un SE, necesariamente pasa por la **explicitación** de sus elementos. Es decir, todo conocimiento utilizado por el SE, que incluye el conocimiento almacenado en la BC y el conocimiento utilizado en su diseño, pero que normalmente no queda codificado (conocimiento no compilado), debe quedar plasmado en forma explícita, en una representación que permita usar este conocimiento para generar explicaciones sobre diferentes aspectos del SE. Incluso, Clancey propone la utilización de una misma BC desde distintos contextos, por ejemplo, una BC puede ser utilizada para resolución de problemas y como sistema de instrucción. La idea se ha concretado a través del sistema GUIDON[5], un tutor acerca del diagnóstico de enfermedades infecciosas a la sangre, que utiliza la misma BC de MYCIN, el sistema que hace el diagnóstico.

La segunda idea nace de la anterior y es que es necesario **separar, diferenciar, clasificar y aun agregar niveles de conocimiento y de meta-conocimiento** (conocimiento acerca de su propio conocimiento) a un SE. Clancey agregó al sistema MYCIN un nivel de "conocimiento estratégico", generando NEOMYCIN[7,10,16]. Swartout propone agregar "conocimiento de justificación" en el momento de creación de la BC, el cual no es usado directamente en las inferencias del programa, pero sí en la generación de las explicaciones[1,3,6]. Hoy, al menos se ha hecho la diferencia entre lo que es **conocimiento del dominio** (hechos y reglas que, independientemente del caso en que se apliquen, siempre son verdaderas con certeza total), y **conocimiento de control** (reglas que, dependiendo del caso de aplicación, modifican y controlan la aplicación de las reglas del dominio, variando también su grado de certeza).

Otra consideración importante acerca de la generación de explicaciones es el compromiso entre generar explicaciones en forma totalmente automática, enfoque propio de las herramientas de construcción de SE, o incorporarlas en forma ad-hoc a un SE ya construido (incorporación manual). La tabla siguiente muestra las características de uno u otro enfoque:

Generación de Explicaciones

generación automática	**incorporación ad-hoc**
• bajo nivel (en términos de reglas y variables)	• mayor abstracción (reglas son usadas como referencia)
• no dependiente del dominio	• dependiente del dominio
• enfoque sintáctico	• enfoque semántico
• más útil al ingeniero del conocimiento	• más útil al usuario final

Tabla 1

La generación automática de explicaciones es enteramente en términos de las reglas y variables de la BC del programa, porque sus mecanismos deben ser independientes de la BC a la cual se aplican. Es así como funcionan los clásicos WHY y HOW.

Por el mismo motivo el enfoque que se adopta es sintáctico (el propio razonamiento lo es), a diferencia de los sistemas manuales en que las variables se usan como referencia para obtener explicaciones más abstractas.

Por el mismo hecho de no depender de la BC a la que se aplica, no depende del dominio de aplicación. Por el contrario, en la incorporación manual de explicaciones, hay **principios** que explican el comportamiento del SE que se aplican en una área de conocimiento específica,. Por ejemplo, es sabido y bastante universal que los médicos evalúan a sus pacientes en una secuencia de preguntas que obedece a una razón económico-práctica: primero la anamnesis, después el examen físico y finalmente los exámenes de laboratorio. Chandrasekaran[3] arguye que estos principios dependen del tipo de resolución de problema que se quiere proveer con el SE, que en el ejemplo dado corresponde al diagnóstico en medicina. Algunos tipos de resolución muy generales son[11]: Clasificación (interpretación de medidas, diagnóstico, búsqueda y corrección de errores, detección de fallas, tutores en línea), Diseño y Síntesis (configuración de sistemas), Asistencia Inteligente, Predicción, Programación de Tareas (Scheduling), Planificación, Monitoreo, Control, etc.

Por el hecho de que las explicaciones en los sistemas automáticos se expresan en términos de las variables y reglas que el motor de inferencia ha utilizado, éstas son más útiles al ingeniero de conocimiento para detección de faltas que para un usuario que consulta el SE. Por el contrario en los sistemas manuales, el objetivo principal de las explicaciones es el convencimiento del usuario final de que las soluciones generadas son correctas.

3. Hipermedio como mecanismo de apoyo a las Explicaciones

3.1. Hipermedio

El término Hipertexto acuñado por Theodor Nelson[9], describe una vasta red de fragmentos de texto unida por relaciones, un sistema de lectura y escritura electrónica que usa el poder del computador para capacidades más poderosas que sólo editar y desplegar información.

El término hipermedio o multimedio (hypermedia) es sólo una extensión del término hipertexto que incorpora otros medios, además del texto. Con un sistema de hipermedios, los autores (en este caso, los propios expertos) pueden crear un cuerpo ligado de materias que incluyen texto, gráfica estática y animada, video y sonido.

Los sistemas de hipermedio e hipertexto permiten al usuario seguir redes complejas de información rápidamente. Los autores pueden así ligar pasajes y referencias, y colapsar o expandir líneas; los usuarios pueden moverse libremente a través de fragmentos de texto para encontrar palabras, referencias bibliográficas, definiciones y pasajes relacionados. El uso de hipermedio posibilita la búsqueda y relación de la información de una forma no-lineal más parecida a cómo el hombre razona.

3.2. El Esquema de Explicaciones Propuesto

Se propone un enfoque de generación de explicaciones basado en Hipermedio que, apoyándose en representaciones de bajo nivel del conocimiento, permita recorrer y navegar libremente por el conocimiento que justifica la BC, pero que no es directamente utilizado en el razonamiento.

La representación de bajo nivel consiste de un grafo de reglas, el cual, durante el razonamiento es recorrido mediante los mecanismos de control clásicos (backward y forward), y durante una consulta explicativa, es recorrido mostrando las hipótesis y subhipótesis que se intentarán probrar (WHY), o las hipótesis y subhipótesis que fueron exitosas o no (HOW).

A cada variable de la BC, se asocian nodos de un grafo virtual, el grafo de hipermedio. Cada nodo de este grafo representa una explicación tendiente a *clarificar* una pregunta determinada (apoya a decidir qué responder al SE), *enseñar* el concepto involucrado y/o *justificar* la lógica del SE (¿por qué el SE hace esta pregunta?, un "*WHY*" más semántico).

La tecnología de hipermedio enfoca el problema de la interfaz humano-computador, aspecto débilmente solucionado desde el paradigma de sistemas expertos, en la generación de explicaciones.

El diseñador del SE, el ingeniero del conocimiento (IC), puede así separar el problema de diseñar el SE como resolutor de problemas (diseño del SE, propiamente tal) y el problema de incorporar y administrar adecuadamente el conocimiento de justificación. Este último se enfoca desde el paradigma de hipermedio, que resuelve los problemas propios del área de interfaces humano-computador, tales como las clásicas preguntas de *¿dónde estoy?*, *¿cómo llegué aquí?* y *¿a dónde puedo ir?*[14], o el estudio del comportamiento del usuario respecto a las partes del conocimiento más y menos frecuentemente consultadas, etc.

Este es un mecanismo intermedio entre la generación automática de explicaciones a partir de la BC y la incorporación ad-hoc de conocimiento de apoyo a las explicaciones, que trata de integrar las ventajas de ambos enfoques.

Hay dos niveles que se proveen:

1. Explicación de la Inferencia a partir de la base de reglas (BC).
2. Justificación de la BC.

La explicación de la inferencia a partir de la BC, se centra en los aspectos locales y específicos de la inferencia, apoyándose en las reglas y variables de la BC. Se proveen los tradicionales WHY y HOW. El comando WHY especifica por qué una determinada respuesta es requerida por el sistema, y el comando HOW, cómo se ha obtenido un determinado estado, pero con un énfasis en la representación gráfica de la BC: se despliega el árbol de historia de un determinado estado, pudiendo el usuario pedir por información de una variable determinada a la manera de los sistemas de hipertexto, es decir, a partir de íconos seleccionables.

El segundo aspecto, más global y abstracto, se refiere a la justificación de la BC: a partir de un estado de consulta en que el SE requiere un dato por parte del usuario, éste puede explorar el concepto que hay detrás. Se ligan partes de la BC (reglas) a conocimiento del dominio que justifica esa porción de la BC, es decir, justifica la pregunta actual que hace el sistema, y ayuda a decidir respecto del sentido y tipo de respuesta que el SE requiere.

Dentro del conocimiento del dominio se diferencian dos tipos de explicación: explicación de conceptos (WHAT) referentes a la pregunta actualmente activa, y justificación respecto de la pregunta, que describe globalmente la estrategia seguida por el SE. Este último tipo de explicación pretende dejar explícito el conocimiento de estrategia utilizado en el diseño de la base de reglas.

El conocimiento del dominio podría ser parte de, por ejemplo, un tutor inteligente[4], el cual utiliza este conocimiento para enseñar, como en la figura 1:

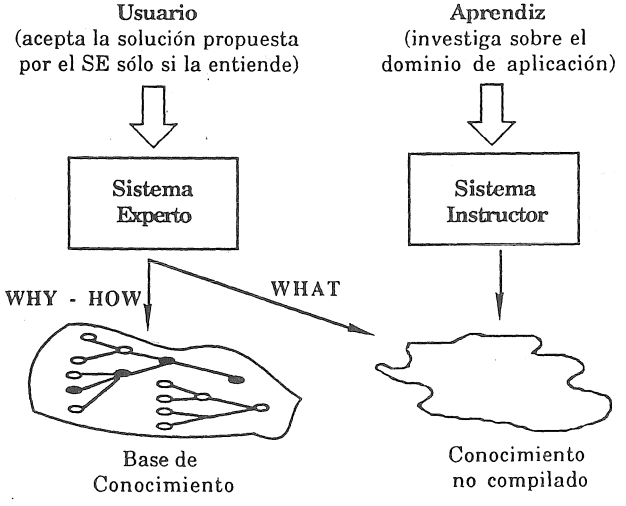


Figura 1. Arquitectura Mixta.

Los mecanismos de bajo nivel (Why-How) sólo proporcionarán el punto de partida para la navegación a través del conocimiento. Proveen las variables y reglas sobre las cuales se requiere una explicación y/o justificación. El mecanismo de hipermedio utiliza estos datos como claves para accesar la porción del conocimiento del dominio que contiene la explicación y/o justificación requerida.

Esta es la principal diferencia con los sistemas tradicionales. Cada punto explicable de la BC puede ser reforzado con conocimiento de apoyo tanto como el experto desee, en un desarrollo incremental. Para adquirir el conocimiento de apoyo se ha separado lo que corresponde a Adquisición de Conocimiento (tendiente a desarrollar el SE como resolutor) de la "Adquisición de Explicaciones", cuyo objetivo es adquirir el conocimiento para explicar, clarificar y justificar las reglas que se generaron en la etapa anterior, además de la definición, por parte del IC y del experto, de los medios más adecuados para presentar cada porción de este conocimiento.

Existe una estructura de datos manipulada en el razonamiento (las reglas y variables), la cual se utiliza como llave de acceso (una variable) al contexto del hipermedio, donde se encuentra el conocimiento de apoyo. Esta estructura es independiente de la BC a la que se aplica, que puede ser reforzada con más niveles de conocimiento para explicación.

La ventaja del esquema propuesto es la independización de ambos sistemas, al menos desde tres puntos de vista:

- Del diseño y desarrollo, porque el IC diseña primeramente el SE como resolutor de problemas, concentrándose en mejorar el desempeño como tal, e independientemente, el IC diseña el SE en cuanto a conocimiento de justificación, concentrándose en los aspectos cognoscitivos y de interfaz humano-computador.

- De la funcionalidad y mantenibilidad, porque esta separación permite alterar la BC de razonamiento (las reglas manejan la información de cómo resolver el problema), que tiene una función de resolución de problemas), alterando ínfimamente la BC de justificación (la base teórico-conceptual no cambia porque se haya cambiado la forma de resolver el problema), esta BC tiene una función de clarificación, convencimiento y pedagogía.

- De la eficiencia en la inferencia, porque el SE no se recarga con conocimiento de justificación que sólo será utilizado con poca frecuencia (sólo cuando el usuario requiere una explicación).

El hipermedio es una herramienta apropiada para lograr la encapsulación del conocimiento de justificación, por la transparencia en el manejo de la información que provee al usuario, y la facilidad de relacionar grandes cantidades de información. El hipermedio permite además, resaltar el caracter pedagógico de un SE (a través de gráfica, animaciones, etc.), del mismo modo como lo podría hacer un experto humano.

4. Prototipos Desarrollados

Existen dos prototipos desarrollados en los que se ha puesto en práctica el esquema presentado.

En el campo médico, un SE de apoyo al Diagnóstico Diferencial del Vértigo[1], cuyos mecanimos de explicación a través de multimedios, principalmente animaciones y gráficos, han sido útiles para mostrar los fenómenos clínicos de los cuales el sistema requiere información. Se proyecta completar el sistema de consulta y dotarle de mayor énfasis en el caracter de entrenamiento, para ser usado por médicos jóvenes. La figura 2 muestra partes de una interacción con el SE:

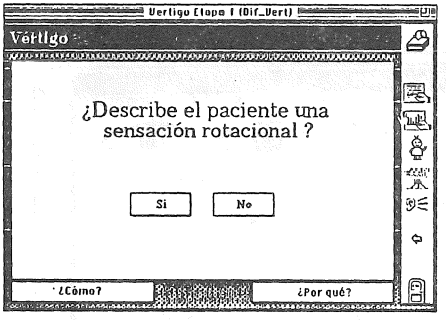


Figura 2a. Esquema básico de pregunta.

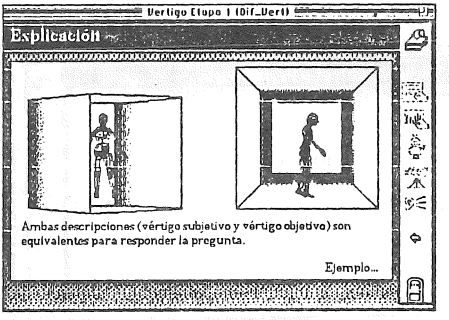


Figura 2b. Toma de una animación que muestra qué se entiende por "sensación rotatoria".

[1] desarrollado por Enrique Hinostroza, Departamento Ciencia de la Computación, Escuela de Ingeniería, Pontificia Universidad Católica de Chile.

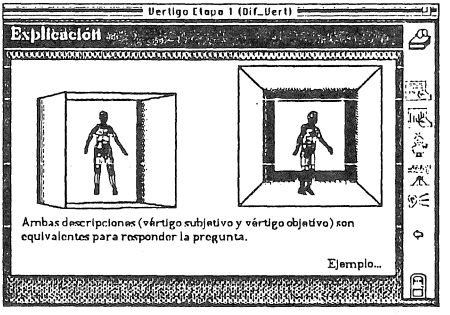


Figura 2c. Otra toma de la misma animación, se hace distinción entre vértigo objetivo y subjetivo.

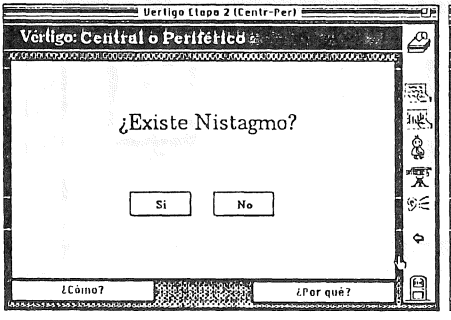


Figura 2d. Las características del nistagmo permiten discernir entre distintos tipos de vértigo.

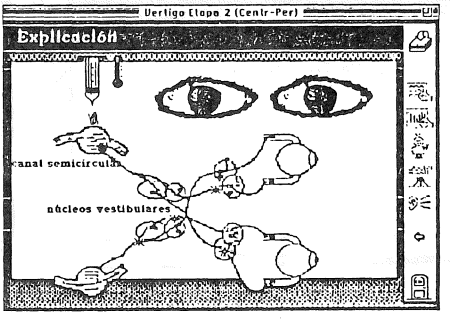


Figura 2e. Parte de una animación mostrando la Prueba Calórica, examen que se aplica a los pacientes para provocar Nistagmo.

Otro sistema, en el campo de la ingeniería de construcción, consiste en un SE de Detección de Fallas en Hormigones[2]. El SE esta provisto de explicaciones animadas acerca de los aspectos físico-mecánicos del hormigón, que ayudan a clarificar lo que el sistema pregunta en un determinado momento y a justificar las preguntas específicas que el sistema hace. La figura 3 muestra un aspecto de este SE:

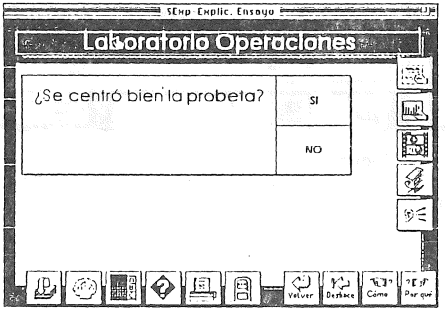


Figura 3a. El centrado de la probeta influye en el resultado del ensayo de compresión.

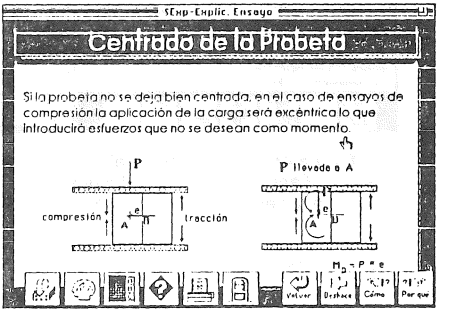


Figura 3b. Explicación gráfica de por qué falla el ensayo de compresión si la probeta está descentrada.

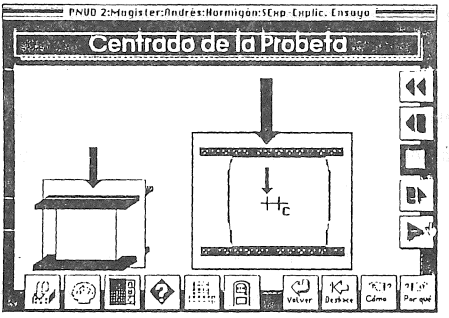


Figura 3c. Parte de una animación mostrando los esfuerzo internos al aplicar el ensayo.

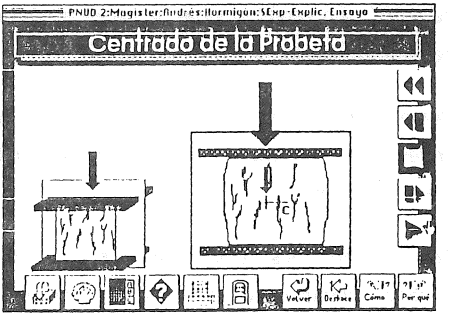


Figura 3d. Casi al final del ensayo, aparecen roturas anormales.

[2] desarrollado por Julián Mardones, Departamento Ciencia de la Computación, Escuela de Ingeniería, Pontificia Universidad Católica de Chile.

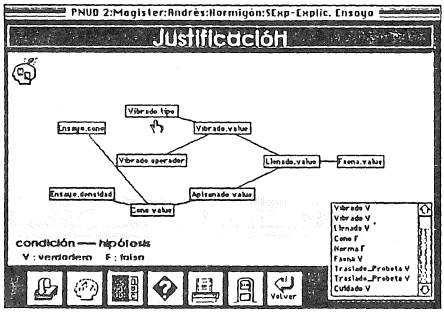


Figura 4a Comando HOW al preguntar por la hipótesis **Faena.value**, el árbol muestra la secuencia de variables gatilladas para probar la hipótesis.

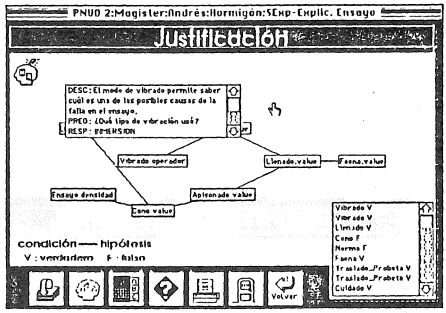


Figura 4b. El usuario pide información respecto de la variable **vibrado.tipo** (figura 4a).

5. Conclusiones

Los requerimientos de un SE para proveer explicaciones consisten en, primeramente, hacer explícitos los niveles y tipos de conocimiento utilizados en este, ya sea en el razonamiento mismo o en la construcción de su BC. En segundo lugar, la clasificación de diferentes niveles de conocimiento es necesaria para generar explicaciones a distintos niveles.

Con estos requerimientos, la BC de un sistema resolutor de problemas se verá recargada con información irrelevante para el razonamiento, pero sí útil en la generación de explicaciones, haciéndola posiblemente menos eficiente.

Por esta razón los sistemas actuales se han enfocado demasiado en mecanismos de explicación locales al motor de inferencias y a la base de conocimiento utilizada en el razonamiento, en este trabajo se propone separar ambos objetivos: una base de conocimiento para razonar, y una base de conocimiento para explicar y justificar la base de conocimiento anterior. Las ventajas de esta separación apuntan a mejorar el desarrollo, la mantenibilidad, la eficiencia, y la funcionalidad del SE.

La utilización de herramientas de hipermedio puede ser una solución apropiada para procurar la suficiente independencia entre el SE, como resolutor de problemas, y el SE, como explicador del conocimiento usado en la razonamiento.

Además el hipermedio permite explotar capacidades gráficas del hardware, haciendo más fidedignos los fenómenos explicados y resaltando el caracter de instructor de los SE´s.

Así como un experto humano puede ser un buen instructor para los novatos, el conocimiento usado para explicar puede ser utilizado para enseñar aspectos del dominio a través de un instructor inteligente. La doble utilidad del mismo conocimiento acorta la distancia que las personas tienen entre el concepto del sistema experto computacional y el concepto del experto humano real.

Agradecimientos

Queremos agradecer a Cecilia Alvarez, Gonzalo Diethelm, Enrique Hinostrosa, Edmundo Leiva, Julián Mardones y Marcelo Miranda, por sus críticas y sugerencias en la preparación de este artículo.

Esta investigación ha sido financiada por el Programa de las Naciones Unidas para el Desarrollo, PNUD, proyecto CHI/80/82 y por el Fondo Nacional de Ciencia y Tecnología, FONDECYT, proyecto 89/0591.

BIBLIOGRAFIA

1. Michael Wick and James Slage, "An Explanation Facility for Today`s Expert Systems", University of Minnesota. IEEE-Expert, Spring 1989.
2. Andrew Basden, "On the application of expert systems", Decision Support Systems Group. United Kingdom. In "Developments in expert systems", edited by M. J. Coombs, Department of Computer Science, University of Strathclyde, Glasgow, Scotland; Academic Press, Inc. 1984.
3. B. Chandrasekaran, M. Tanner and J. Josephson, "Explaining Control Strategies in Problem Solving", IEEE-Expert, Spring 1989.
4. Heinz Mandl, Alan Lesgold, "Learning Issues for Intelligent Tutoring Systems". Cognitive Science, edited by Springer-Verlag New York Inc., 1988.
5. D. Sleeman and J. S. Brown, "Intelligent Tutoring Systems", Academic Press Inc., 1982.
6. Peter Jackson, "Introduction to Expert Systems". Addison-Wesley Publishing Company, Inc. . 1986.
7. D. W. Hasling, W. J. Clancey and G. Rennels, "Strategic explanations for a diagnostic consultation system", in "Developments in experts systems", reference 2.
8. Bruce G. Buchanan, Edward H. Shortliffe, "RULE-BASED EXPERT SYSTEMS, THE MYCIN EXPERIMENTS OF THE STANFORD HEURISTIC PROGRAMMING PROYECT". Addison-Wesley Publishing Company, Inc. , 1984.
9. Gary Marchionini and Ben Shneiderman, "Finding Facts vs. Browsing Knowledge in Hypertext Systems". University of Maryland. COMPUTER, January 1988.
10. Claudia Jiménez D., "An Architecture for Explanation in Expert Systems", COGNITIVA 90, vol.2, Madrid, España. 20-23 Nov: 1990.

11. William B. Gevarter, "The Nature and Evaluation of Commercial Expert System Building Tools". NASA Ames Research Center. COMPUTER. May 1987.

12. Vasant Dhar and Harry E. Pople, "Rule-Based versus Structure-Based Models for Explaining and Generating Expert Behaviour", Communications of the ACM, June 1987. Vol. 30, Number 6.

13. Michael Wick and James Slage, "The Partitioned Support Network for Expert System Justification", IEEE-Transactions on Systems, Man and Cybernetics. May/June 1989. Vol. 19, number 3.

14. Jakob Nielsen, "The Art of Navigating through HYPERTEXT", Communications of the ACM, March 1990. Vol. 33, Number 3.

15 Avron Barr, Paul Cohen, Edward Feigenbaum, "The Handbook of ARTIFICIAL INTELLIGENCE", volume IV. Addison-Wesley Publishing Company, Inc. December 1989.

16 Suhayya Abu-Hakima and Franz Oppacher, "Improving explanations in knowledge-based systems: RATIONALE", Knowledge Acquisition, volume 2, number 4, December 1990, pp. 301-343. Academic Press.

Implementación de un Algoritmo de Puntos Interiores para Programación Lineal

Marielba Rojas
Depto. de Computación
y Tecnología de la Información

Marianela Lentini
Depto. de Matemáticas
Puras y Aplicadas

Universidad Simón Bolívar
Apartado 89000, Caracas 1080-A, Venezuela
sun!emsca!usb!marielba@sun.com
sun!emsca!usb!mlentini@sun.com

Resumen

Se presentarán consideraciones numéricas sobre la implementación de un Algoritmo Primal de Reducción de Potencial para programación lineal con el procedimiento para actualización de cotas inferiores original de Clovis C. Gonzaga.

Las consideraciones incluyen la resolución de sistemas de ecuaciones esparcidas, simétricas y positivas definidas de gran escala, mediante el método de Gradiente Conjugado Precondicionado, y un enfoque novedoso en el cálculo de un precondicionador para este método.

1 Introducción

Consideremos el problema de programación lineal en su forma estándar

$$P: \quad min\ c^T x \tag{1}$$

$$s.a.\ Ax = b \tag{2}$$

$$x \geq 0 \tag{3}$$

donde $A \in \mathcal{R}^{m \times n}, c \in \mathcal{R}^n, b \in \mathcal{R}^m, m < n$.

Las técnicas para resolver este problema incluyen:

- El Método Simplex (1946) de George Dantzig [3], con complejidad exponencial en el peor caso pero con comportamiento lineal en la práctica. El método avanza por puntos en la frontera del conjunto factible.
- El Método de Elipsoides (1978) de Khachiyan [12] con complejidad de $O(n^4 L)$, pero ineficiente en casos prácticos. Con este método se demostró que el problema de programación lineal estaba en P. La estrategia utilizada involucra la construcción de elipsoides que se van reduciendo hasta "encerrar" el óptimo.
- Los Métodos de Puntos Interiores, desarrollados desde la aparición del Algoritmo de Karmarkar en 1984 [10] con complejidad polinomial con órdenes entre $O(n^3 L)$ y $O(n^{3.5} L)$. Estos métodos han mostrado un excelente desempeño práctico, especialmente para problemas de gran escala. A diferencia del Simplex, estos métodos avanzan por puntos interiores del conjunto factible que convergen al óptimo.

En este trabajo describiremos los detalles relativos a la implementación numérica de un método de puntos interiores para programación lineal . Otros trabajos que enfocan los problemas de implementación subyacentes a estos algoritmos son los de Adler et al. [1], [2]; McShane et al. [15]; Monma y Morton [17]; Marsten et al. [14]; Mehrotra [16]; Gill et al. [6]; entre los más representativos.

El método seleccionado se clasifica como un algoritmo Primal de Reducción de Potencial, el mismo fue presentado por Gonzaga en [9]; en este método sólo se almacenan las variables primales, a diferencia de otros métodos que almacenan o bien las variables duales (métodos duales) o ambos grupos de variables (métodos primales–duales).

2 Algoritmo Numérico

El algoritmo que presentamos es el resultado de reformular el algoritmo de Gonzaga dado en [9], el cual reduce en cada paso la siguiente función llamada función potencial

$$x > 0 \longmapsto f_v(x) = q\log(c^T x - v) - \sum_{i=1}^{n} \log x_i$$

donde v es una cota inferior para el costo óptimo y $q \geq n$ es constante.

Se harán las siguientes hipótesis:

Si $S = \{x/Ax = b \ y \ x \geq 0\}$ (conjunto factible) y $S^\circ = \{x/Ax = b \ y \ x > 0\}$ (interior relativo de S), entonces

(i) A es de rango completo.

(ii) S es acotado.

(iii) S° es no vacío.

(iv) P tiene una solución óptima $\hat{x}$ con costo $\hat{v}$.

Algoritmo Práctico de Reducción de Potencial

Dados $q > n$, $v_\circ < \hat{v}$, $x^\circ \in S^\circ$

$k := 0$

REPEAT

Cambio de Escala: sea $X_k = diag(x_1^k, x_2^k, \ldots, x_n^k)$

$x^k = X_k e$

$\overline{f}_v(e) = q\log(\overline{c}^T e - v_k)$

$\overline{A} = AX_k$

$\overline{c} := X_k c$

Calculos Relativos a la Proyeccion:

$(AX_k^2 A^T)u = AX_k\overline{c}$

$c_p := \overline{c} - X_k A^T u$

$(AX_k^2 A^T)w = Ax^k$

$e_p := e - X_k A^T w$

$\alpha := \frac{e-e_p}{\|e-e_p\|^2}$

Cota Inferior:

$v := \bar{c}^T e - c_p^T + sup\{\mu / c_p - \mu a \geq 0\}$

Direccion: usar una de las formulas siguientes

(i) Cauchy ($q > n$): $h := -\frac{q}{c^T x^k - v} c_p + e_p$

(ii) Karmarkar ($q = n$): $h := -c_p + \frac{\bar{c}(e-e_p) - v_{k+1}}{\|e-e_p\|^2} e_p$

Busqueda Lineal:

$\overline{\lambda} := argmin\{\overline{f}_v(e + \lambda h) / e + \lambda h > 0,\ \lambda > 0\}$

Calculo del nuevo punto factible:

$y := e + \overline{\lambda} h$

Retorno al espacio original:

$x^{k+1} := X_k y$

UNTIL $\frac{|c^T x^k - v|}{|c^T x^k|} < \epsilon$

Donde e_p, c_p son las proyecciones de e y de $\bar{c}$ sobre el espacio nulo de $\overline{A} = AX_k$, respectivamente.

Como puede observarse hay dos direcciones posibles para reducir la función objetivo: la dirección de Cauchy, que da origen a los algoritmos llamados afines y la de Karmarkar, que origina el algoritmo del mismo nombre. Ambas direcciones se obtienen combinando una dirección de minimización del costo y una dirección de centralización o alejamiento de la frontera de S.

En cuanto al cambio de escala, este se realiza en forma implícita en todos los casos, salvo para el vector c para el cual se realiza el cálculo efectivo de $\bar{c}$.

Bajo las hipótesis establecidas, este algoritmo tiene una complejidad de $O(n^{3.5}L)$ operaciones en el peor caso, donde L es el número total de bits necesarios para representar los datos del problema.

De las operaciones que realiza el algoritmo, la más costosa es la asociada a la proyección, pues requiere de la resolución de dos sistemas de ecuaciones, cuya matriz es

$$D_k = AX_k^2 A^T.$$

Además, la matriz de estos sistemas se obtiene como un producto de matrices y esta

operación también es costosa; en las implementaciones debe evitarse el cálculo de este producto. Las implementaciones eficientes [1], [17] conocidas hasta el presente, almacenan la parte invariante del producto (asociada a los elementos de A), para reducir los cálculos; sin embargo, esta estrategia es costosa en almacenamiento.

En razón de lo anterior en la implementación que se describe en este trabajo, se le asignó mayor prioridad a la búsqueda de una alternativa eficiente tanto para resolver los sistemas como para evitar el costo en almacenamiento.

En las siguientes subsecciones describiremos las técnicas y algoritmos utilizados para implementar los siguientes aspectos del algoritmo:

- Proyección.
- Búsqueda lineal.
- Cálculo del punto inicial.
- Cálculo de una cota inferior inicial para $\hat{v}$.
- Escogencia del parámetro q.
- Criterio de parada.

2.1 Proyección

La matriz de los sistemas de ecuaciones involucrados en el cálculo de la proyección, tiene las siguientes características:

- Es simétrica.
- Es positiva definida.
- Es esparcidacon un grado de esparcidad en muchos casos similar al de A, que puede llegar a un 99% para la mayoría de las aplicaciones.

Un método que ha dado excelentes resultados en problemas esparcidade gran tamaño es el Método de Gradiente Conjugado Precondicionado (ver [7] para un excelente estudio

del método), por lo cual se decidió utilizarlo en esta aplicación. No obstante, el método presenta el inconveniente de que requiere del cálculo de un precondicionador para ser eficiente.

Un precondicionador ampliamente utilizado es el factor de Cholesky "incompleto", $\tilde{L}$, que se obtiene al aplicar el Método de Cholesky sin calcular los elementos en posiciones donde hay ceros en la matriz original del sistema para evitar el *fill-in* es decir, la aparición de elementos no nulos en posiciones donde había ceros.

En esta aplicación se utiliza un factor de Cholesky "incompleto" como precondicionador, pero el mismo no se calcula con procedimientos convencionales pues eso implicaría calcular y almacenar D_k o ciertos productos externos (ver [1], [14], [16], [6]).

En lugar de usar el Método de Cholesky directamente sobre D_k, aplicamos transformaciones de Householder para econtrar la descomposición QR de la matriz $B = (AX_k)^T$, es decir calculamos Q ortogonal y R triangular superior tales que $B = QR$. Es fácil observar que R^T es el factor de Cholesky de B^TB. De esta manera *no es necesario* almacenar D_k, sólo se guarda A en formato *sparse*.

Al aplicar la factorización QR (con pivoteo por razones de estabilidad numérica) podría generarse *fill-in* por lo cual en la implementación se sigue una estrategia análoga a la empleada en la factorización incompleta de Cholesky. El factor $\tilde{L}$ así obtenido se utiliza para resolver sistemas triangulares en las iteraciones de Gradiente Conjugado, en razón de lo cual la estabilidad de esos sistemas es de suma importancia. Manteuffel [13] identificó un conjunto de matrices para los cuales el factor incompleto de Cholesky conduce a sistemas estables; es necesario realizar un estudio similar para nuestra estrategia.

Es bien conocido que el *fill-in* en los procedimientos de eliminación depende del orden en que aparecen las filas de la matriz de coeficientes. En nuestro caso cabe esperar que un reordenamiento de las filas y columnas mejoren el comportamiento del precondicionador, tal como lo anuncian Duff y Meurant en [5]. La mayoria de las implementaciones mencionadas utilizan técnicas de reordenamiento con excelentes resultados.

2.2 Búsqueda Lineal

Una vez obtenida la dirección de minimización h es necesario calcular la longitud del paso a tomar sobre esa dirección para reducir la función objetivo, al menos en una cantidad que garantice los resultados teóricos. Nuestra decisión en este sentido fue utilizar el mismo criterio en el que se basan dichos resultados teóricos: se escoge como paso sobre h la distancia del punto factible actual a la frontera del conjunto factible.

Este cálculo produce un punto en la frontera de S y el algoritmo presentado trabaja sobre puntos en el interior de S; por esta razón, una vez calculada la distancia a la frontera, se toma un porcentaje (0.9995 como sugieren McShane et al. [15]) de esa distancia para mantener el iterado en S°. Si bien esta estrategia no es muy elegante, ha dado excelentes resultados en la práctica.

2.3 Cálculo del punto inicial

El algoritmo supone que se conoce un punto interior inicial. Las implementaciones deben proveer algún mecanismo para calcular este punto interior. En nuestro caso se utiliza un método clásico fase I- fase II.

En la fase I se aplica el algoritmo a un problema modificado P', para el cual se conoce un punto interior inicial, y tal que una solución óptima del mismo es un punto interior inicial para el problema original. En base a ese punto, la fase II aplica el algoritmo al problema original.

El problema modificado es el siguiente

$$
\begin{aligned}
P' : \quad & \\
& min\ c^T x + c_a x_a \\
& s.a.\ Ax + (b - A\tilde{x}^\circ)x_a = b \\
& x, x_a \geq 0
\end{aligned}
$$

Para cualquier $\tilde{x}^\circ > 0$ y $x_a = 1$, el vector $< \tilde{x}^\circ, x_a >$ es un punto interior para P'. La fase I del algoritmo debe minimizar x_a, por lo que el costo asociado, c_a, debe ser mayor

que los otros costos. En nuestra implementación tomamos

$$c_a = n^2 * \max_{1 \le i \le n} |c_i|$$

Para determinar la terminación de la fase I se emplea una tolerancia de factibilidad $\gamma \approx 10^{-3}$ que se interpreta de la siguiente forma:

(i) Si $|x_a| < \gamma$, entonces terminó la fase I y el último iterado es un punto interior para P.

(ii) Si se cumple el criterio de parada y $x_a < -\gamma$, entonces el problema no tiene solución acotada.

(iii) Si se cumple el criterio de parada y $x_a \ge \gamma$, entonces el problema no tiene solución factible.

Una elección obvia para x^0 es el punto e, pero también puede usarse cualquier vector que satisfaga la restricción de positividad, en este sentido puede consultarse [15].

2.4 Cálculo de una cota inferior inicial v_0 para $\hat{v}$

Para resolver este problema hay al menos dos opciones, una es calcular la cota inicial usando el procedimiento de Todd y Burrel [18], que implica resolver un sistema de ecuaciones; la otra opción es tomar $v_0 = -\infty$ lo que es válido si el procedimiento de actualización es eficiente, como en el caso del procedimiento de Gonzaga [9]. La decisión en este sentido fue la de tomar v_0 como el menor valor representable en punto flotante de doble precisión.

2.5 Escogencia del parámetro q

Los resultados teóricos [8] que establecen polinomialidad para algoritmos afines requieren que $q \ge n + \sqrt{n}$, $q = O(n)$; el algoritmo de Karmarkar requiere que $q = n$.

Decidimos utilizar el valor de $q = K * (n + \sqrt{n})$, para la versión afín polinomial del algoritmo, y $q = n$ para el algoritmo de Karmarkar. La constante K se utiliza para dar

mayor peso a la componente de minimización del costo en la dirección de reducción. En la actualidad se utiliza $K = \max_{1 \leq i \leq n} |c_i|$.

2.6 Criterio de Parada

En lugar de usar una precisión de 2^{-L} como en los resultados teóricos se escoge una precisión de $\epsilon \approx 10^{-8}$, pero también puede usarse el ϵ de la máquina.

Puede ocurrir que por problemas de precisión el criterio anterior no llegue a satisfacerse, por lo que se require de un criterio adicional que en nuestro caso consiste en detectar un incremento en la función objetivo.

3 Resultados

En la actualidad se realizan experimentos sobre datos de prueba obtenidos a través de *netlib* [4] con una implementación desarrollada en lenguaje C [11] sobre un econsiste en SUN [1] Sparc 1+ bajo Unix [2]. En la exposición se presentarán resultados numéricos sobre dichos datos.

4 Conclusiones

De la experiencia obtenida en este trabajo, se ha determinado que es posible realizar implementaciones eficientes de los algoritmos de puntos interiores para programación lineal , siempre que se tome en cuenta a cabalidad las características del problema para poder lograr un equilibrio entre utilización de memoria y número de operaciones.

Lo anterior se aplica en particular a la resolución de sistemas de ecuaciones esparcida, simétricas y positivas definidas que los mencionados algoritmos contemplan. En este sentido, se considera muy prometedor el enfoque seguido en este trabajo, especialmente si se utilizan técnicas de reordenamiento y más aún si se logra incorporar el uso del paralelismo en estas operaciones.

[1] SUN es marca registrada de SUN Microsystems Inc.

[2] Unix es marca registrada de AT&T

Cabe destacar que existen otros aspectos de implementación que deben ser analizados en detalle pues pueden contribuir a mejorar sustancialmente la eficiencia de la misma. Entre estos tenemos: el cálculo de c_a en la fase I, el procedimiento para actualizar la cota inferior y el cálculo de q.

Referencias

[1] I. Adler, N. Karmarkar, M. Resende, y G. Veiga. Data structures and programming techniques for the implementation of Karmarkar's algorithm. *ORSA Journal on Computing*, 1(2), Abril 1989.

[2] I. Adler, N. Karmarkar, M. Resende, y G. Veiga. An implementation of Karmarkar's algorithm for linear programming. *Mathematical Programming*, 44, 1989.

[3] G. Dantzig. *Linear Programming and Extensions.* Princeton University Press, New Jersey, 1963.

[4] J. Dongarra y E. Grosse. Distribution of mathematical software via electronic mail. *Communications of the ACM*, 30(5), 1987.

[5] I. Duff y G. Meurant. The effect of ordering on Preconditioned Conjugate Gradients. *BIT*, 29, 1989.

[6] P. Gill, W. Murray, J. Tomlin, y M. Wright. On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method. *Mathematical Programming*, 36, 1986.

[7] G. Golub y Ch. Van Loan. *Matrix Computations.* The John Hopkins University Press, Baltimore, 2ª edición, 1989.

[8] C. Gonzaga. Polynomial affine algorithm for linear programming. Reporte Técnico, COPPE-UFRJ, Río de Janeiro, Enero 1988.

[9] C. Gonzaga. On lower bounds updates in primal potential reduction algorithm for linear programming. Reporte Técnico, COPPE-UFRJ, Río de Janeiro, Junio 1990.

[10] N. Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica*, 4, 1984.

[11] B. Kernighan y D. Ritchie. *The C Programming Language.* Prentice–Hall, Inc., New Jersey, 1978.

[12] L. Khachiyan. A polynomial algorithm for linear programming. *Soviet Mathematics Doklady*, 20, 1979.

[13] T. Manteuffel. Shifted incomplete cholesky factorization. In I. Duff y G. Stewart, editores, *Sparse Matrices Symposium, 1978.* Publicaciones de SIAM , 1979.

[14] R. Marsten, M. Saltzman, D. Shanno, G. Pierce, y J. Ballintijn. Implementation of a dual affine interior point algorithm for linear programming. *ORSA Journal on Computing*, 1(4), 1989.

[15] K. McShane, C. Monma, y D. Shanno. An implementation of a primal dual interior point method for linear programming. *ORSA Journal on Computing*, 1(2), 1989.

[16] S. Mehrotra. Implementations of affine scaling methods: Approximate solutions of systems of linear equations using Preconditioned Conjugate Gradient Methods. Reporte Técnico 89-04, Department of Industrial Engineering and Management Sciences, Northwestern University, Agosto 1989.

[17] C. Monma y A. Morton. Computational experience with a dual affine variant of Karmarkar's method for linear programming. *Operations Research Letters*, 6, 1987.

[18] M. Todd y B. Burrell. An extension of Karmarkar's algorithm for linear programming using dual variables. *Algorithmica*, 1, 1986.

HERRAMIENTAS Y METODOLOGIA PARA EL DESARROLLO DE APLICACIONES ACADEMICAS

Ing. **Adelaide Bianchini T.** Ing. **Edna Ruckhaus de Millán,** M.C.I.S

Oficina Metrópolis
Universidad Metropolitana
Fax: (02) 241.95.75
Dist. Universidad, Autopista Petare-Guarenas
P.O. BOX 76.819 - CARACAS 1070

RESUMEN

El papel de la computadora en el mejoramiento del proceso enseñanza-aprendizaje ha sido estudiado desde la década de los 50 y, cuatro décadas más tarde no está claro qué herramientas y metodologías utilizar para el desarrollo de aplicaciones académicas. El presente artículo describe las experiencias realizadas en el desarrollo de aplicaciones académicas, haciendo énfasis en las herramientas y metodología utilizada. Se dan conceptos relativos a los tipos de aplicaciones académicas, tipos de herramientas y recursos de presentación. Se propone una metodología para el desarrollo de aplicaciones académicas. Luego se presenta una relación conceptual entre estos tres elementos. Por último se describen las herramientas y aplicaciones que se han desarrollado en la Universidad Metropolitana y la evolución de la metodología utilizada.

INTRODUCCION.

El papel de la computadora en el mejoramiento del proceso enseñanza-aprendizaje ha sido estudiado desde la década de los 50 y, cuatro décadas más tarde no está claro qué herramientas y metodologías utilizar para el desarrollo de aplicaciones académicas. El presente artículo describe las experiencias realizadas en el desarrollo de aplicaciones académicas, haciendo énfasis en las herramientas y metodología utilizada.

Estas experiencias se llevan a cabo en la Universidad Metropolitana, a través de la Oficina Metrópolis, que a lo largo de sus cinco años de funcionamiento ha venido trabajando conjuntamente con los profesores y estudiantes de la Universidad, en el desarrollo de aplicaciones académicas utilizando diferentes tipos de herramientas y una metodología que se describirán a lo largo del artículo.

CONCEPTOS.

A continuación se darán algunos conceptos generales que serán utilizados en el presente trabajo.

- Tipos de aplicaciones académicas.

• ***Drill*** **o Práctica**

Consiste en un grupo de ejercicios que se le presenta al estudiante para ser evaluados como "correcto" o "incorrecto". Cuando se utiliza el *Drill* se supone que el contenido del tema sobre el cual tratan los ejercicios ha sido discutido previamente en clase. El tipo de ejercicio presentado al estudiante puede ser de escogencia múltiple, llenar espacios en blanco, o simplemente escribir la respuesta. Si el estudiante da una respuesta incorrecta se le puede dejar que intente nuevamente o también darle la respuesta correcta despues de uno o más intentos.

Al estudiante se le puede dar a escoger entre varios niveles de complejidad y el orden de presentación de ejercicios correpondientes a determinado nivel puede ser aleatorio.

• **Simulación**

Es un modelo dinámico que describe el funcionamiento de un proceso o situación. La computadora ha sido ampliamente utilizada en este área. Los juegos son programas de

simulación, se puede simular el control de una gran empresa y en el área educativa los programas de simulación se utilizan especialmente para la conducción de experimentos de laboratorio. El programa utiliza gráficos y animación para visualizar el experimento y puede dejar que el estudiante participe en la conducción del mismo a su propio ritmo de trabajo. A medida que se va desarrollando el experimento se puede evaluar los conceptos relacionados a través de ejercicios. Un tipo de simulación que se utiliza ampliamente es la **Visualización** de un concepto, en donde una información presentada, generalmente un gráfico, varía según los datos ingresados por el estudiante.

- **Tutorial**

Presenta una explicación de un tema u objetivo el cual es evaluado continuamente ajustándose al ritmo de progreso del estudiante. En este tipo de programas, muchas veces se supone que el estudiante no conoce el concepto con anterioridad, por lo que siempre se le da una explicación antes de darle algún ejercicio. Una variante de este tipo de aplicación es la **Guía de Estudios** en donde se le presentan al estudiante varios conceptos y ejemplos relacionados entre sí.

Cabe aclarar que en muchos casos una aplicación no puede ser catalogada como determinado tipo sino que es una mezcla de varios. Así es común tener un *Drill* que antes de mostrar los ejercicios da un Tutorial del concepto, o un Tutorial que explica el concepto a través de una Simulación.

- Tipos de herramientas.

- **Lenguajes de programación**

Los lenguajes de programación de tercera generación (Pascal, C, Basic) fueron las primeras herramientas utilizadas en el desarrollo de aplicaciones académicas. Aún son ampliamente utilizados dada su flexibilidad para abarcar cualquier tipo de problema.

- **Paquetes de Autor**

Son herramientas elaboradas espècialmente para que profesores desarrollen aplicaciones académicas. Algunos de estos paquetes proveen un lenguaje de programación muy sencillo. En otros la especificación de la secuencia del curso se hace gráficamente a través

- Para el desarrollo de un *Drill* se puede utilizar un Paquete de Autor cuando la **presentación** al usuario no es primordial. En este caso se puede desarrollar rápidamente una aplicación que presente un grupo de ejercicios en orden aleatorio y que incluya mecanismos para la evaluación del estudiante. Si la **presentación** es importante, (por ejemplo, si el usuario es un niño) es preferible utilizar un Sistema de *Hypermedia* aunque el tiempo de desarrollo sea mayor.
- Para el desarrollo de una Simulación se puede utilizar un Paquete de Autor cuando la animación no es muy elaborada. En caso de que sí lo sea se debe utilizar un Sistema de *Hypermedia* que permita incorporar animación.
- Para el desarrollo de un Tutorial, si lo que se va a explicar es un concepto muy específico se puede utilizar el Paquete de Autor, si en cambio se quiere presentar una serie de conceptos relacionados entre sí es preferible utilizar el Sistema de *Hypermedia* ya que está basado en una organización de la información de manera "no lineal".
- Los lenguajes de programación deben utilizarse cuando se requieren elementos que no se encuentran en los lenguajes de un Sistema de *Hypermedia* y Paquete de Autor, como es el caso de estructuras de datos, gran volumen de información y eficiencia de cálculo.

Los requerimientos de presentación, manejo y almacenamiento de información, procesamiento y cálculo de la aplicación a desarrollar, determinan la flexibilidad que se necesita de la herramienta. Por otra parte, para aprovechar al máximo las opciones de un tipo de herramienta, implica que se debe profundizar en su conocimiento, lo cual conlleva a profundizar también en conceptos avanzados de programación al igual que en las características del *hardware* utilizado. Más aún, si se solventan los problemas anteriores, mientras el tipo de herramienta sea más flexible, más tiempo se debe invertir en el desarrollo.

METODOLOGIA.

En toda metodología para desarrollo de sistemas de información generales, se identifican distintos elementos y se consideran varias fases de desarrollo. En términos generales se pueden nombrar los siguientes:

- Elementos básicos.

- El objetivo que se persigue con el sistema (tangible y concreto).

- Los recursos humanos disponibles: Analistas de sistemas, analistas-programadores, programadores, analistas de organización y métodos, etc.
- Los medios y recursos de *software* y *hardware* que se utilizarán.
- El usuario final.

- **Fases.**
  - Análisis de la situación actual y generación de recomendaciones.
  - Diseño del sistema propuesto.
  - Desarrollo (programación).
  - Implantación y pruebas, etc.

En el caso de aplicaciones académicas, que se consideran también sistemas, la metodología utilizada se mueve en una dimensión distinta a las metodologías para el desarrollo de sistemas de información generales, ya que el alcance de los objetivos que se persiguen no es necesariamente concreto, medible y tangible. En los sistemas de información generales, la participación del usuario final en las fases de análisis y diseño es lógica e imprescindible, en cambio en las aplicaciones académicas, lo más probable es que el usuario final participe muy poco en el análisis del problema planteado y nada en el diseño.

En este punto se debe definir quién se cataloga como usuario final, y para esto se considera que el proceso de enseñanza no está desligado del proceso de aprendizaje. Para un profesor-educador, su necesidad es **enseñar**, por lo tanto cuando se desarrolle una aplicación bajo los requerimientos de "el profesor", éste se convierte en el usuario final y él decide si la forma y contenido se adecúan a sus experiencias, estilo y enfoque de enseñanza, etc.; sin embargo, si se toma el punto de vista del estudiante, que es el que desea **aprender**, es éste el verdadero usuario final.

Tradicionalmente, los sistemas de información generales se diseñan y desarrollan para resolver una situación actual que se desea mejorar en forma concreta (se identifican problemas y se dan recomendaciones); en cambio cuando se analiza la "problemática" de un tema que se desea enseñar, el que conoce el problema no necesariamente es el profesor-educador del tema, sino el estudiante que no sabe como enfrentar ciertos conceptos que a él

debe contener todo lo referente a los dos ambientes: el de enseñanza, el cual definirá las secuencias que se considera seguirá el estudiante, y el de aprendizaje que definirá las otras secuencias posibles que puede tomar el estudiante al explorar la aplicación, y las necesidades descritas en el cuadro anterior. Esto significa que se debe observar a través de dos filtros muy distintos, que son la actitud docente y la expectativa del estudiante. Sólo con ésto se puede asegurar tener más de medio camino recorrido, ya que ambos puntos de vista se enfrentan en esta fase, donde la no existencia de un "acuerdo" entre ambas partes, desecharía *a priori* la utilización de la aplicación en el futuro. Hay que considerar también que el grupo de trabajo se enriquece enormemente con la existencia de un experto en técnicas de enseñanza (procesos cognoscitivos).

- **Desarrollo** (programación y ensamblaje de recursos de presentación), es donde se escoge la herramienta más productiva para lograr que el diseño anterior tome cuerpo. Se realiza toda la evaluación y recopilación de recursos (*Multimedia*) que se requieran (dibujos, textos, sonido, efectos especiales, estilos de presentación, formatos, etc.), y se ensambla el "guión". En esta fase el grupo de trabajo debe contar con la asesoría del experto en la herramienta que se decida utilizar.

- **Implantación y pruebas,** se utiliza la primera versión con un grupo piloto de estudiantes y profesores, y como resultado se debe obtener, información adecuada, considerada como una retroalimentación acerca de la aplicación desde el punto de vista de forma, fondo, motivación, etc.. Más aún, esta retroalimentación servirá para definir aquellas secuencias no contempladas bajo el enfoque de enseñanza.

De todo lo anteriormente expuesto, se deduce que las fases de la metodología utilizada en sistemas de información generales son generalmente secuenciales; en cambio con las aplicaciones académicas, existe una constante iteración entre las fases de implantación, desarrollo y diseño.

EVOLUCION DEL DESARROLLO DE APLICACIONES EN LA UNIVERSIDAD METROPOLITANA.

La Universidad Metropolitana a través de la Oficina Metrópolis ha desarrollado una serie de aplicaciones en ambiente Macintosh™ desde sus comienzos en el año 1986. La evolución en cuanto a herramientas, aplicaciones y metodología desarrolladas ha sido la siguiente:

Herramientas y aplicaciones.

Las únicas herramientas de desarrollo disponibles en el año 1986 eran los lenguajes de programación y la primera aplicación desarrollada por la Oficina Metrópolis fue un *Drill* denominado Enlaces Químicos [Theo87] (ver Fig. 1), realizado en *Pascal*. Paralelamente a esta aplicación se desarrolló una Tabla Periódica [Rami&Theo87].

Varios profesores del Departamento de Física y de la Escuela de Ingeniería Eléctrica comenzaron a desarrollar aplicaciones en sus respectivas áreas utilizando lenguajes como *Pascal* y *Basic* . Estas aplicaciones se pueden considerar como una Visualización de determinados conceptos y entre los primeros desarrollos se tienen "Vectores" [Gian86], "Mínimos Cuadrados" [Gian87], "Análisis de Respuesta en Frecuencia" [Hrus87] y "Análisis de Flujo de Carga" [Hrus87]. Actualmente, estos profesores continúan desarrollando aplicaciones utilizando versiones mejoradas de los lenguajes de programación, y utilizando paquetes que aceleran el diseño de un programa al generar un esqueleto del mismo, en base a los recursos de la interfaz que se indiquen tales como "menús" y "ventanas". Entre las aplicaciones desarrolladas se tienen "Simulación de las Prácticas del Laboratorio de Física" [Gian90] (ver Fig. 2), entre otros. También varios tesistas de diferentes Escuelas de Ingeniería han desarrollado aplicaciones utilizando Lenguajes de Programación como herramienta, algunos son "Mac370: Emulador de Assembler" [Lar&Nuñ89], "ODE: Resolución de Ecuaciones Diferenciales Ordinarias" [Gom&Pin&Lop88], en la carrera de Ingeniería de Sistemas, "PerfoPlan" [Rint87] en Ingeniería Civil, "MacEduca" [Stre88], etc.

Hoy en día la Oficina Metrópolis no utiliza los lenguajes de programación para el desarrollo de aplicaciones sino para herramientas de uso general. Entre éstas se encuentran "Manejador de Archivos Indexados VSAM" [Mene&Rami88] y "FuncLib:Librería de Rutinas Matemáticas" [Rami87].

Los primeros *Drill* fueron realizados por tesistas en el área de Idiomas utilizando un Paquete de Autor denominado *Drill* que solo maneja texto. En el año 1987 salieron al

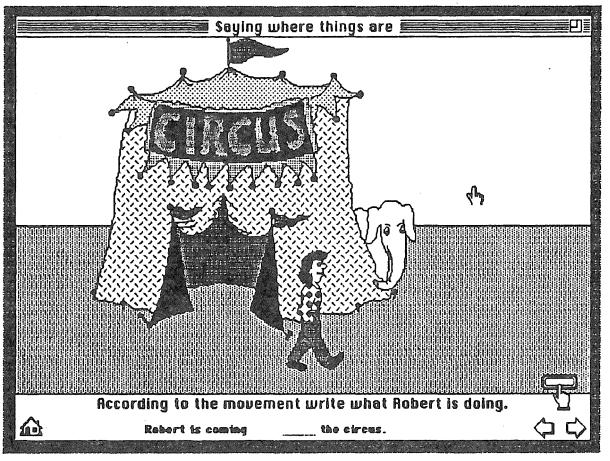


Fig. 4

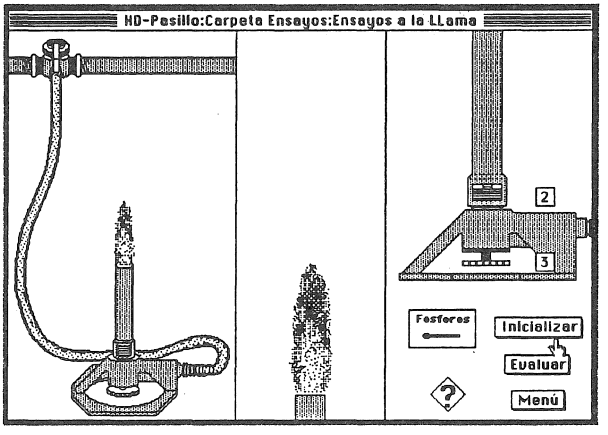


Fig. 5

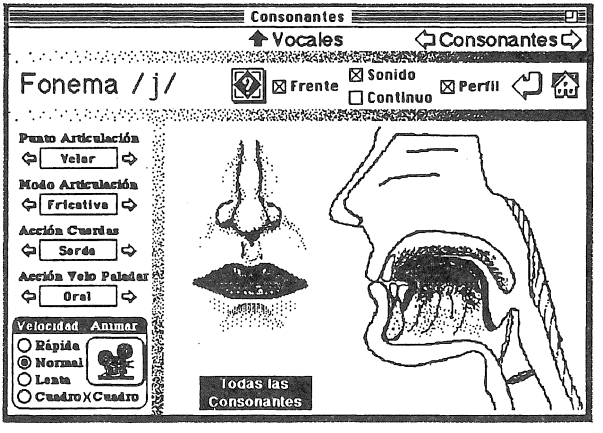


Fig. 6

Metodología.

La metodología utilizada en los desarrollos de la Oficina Metrópolis de la Universidad Metropolitana ha ido evolucionando de la siguiente forma:

- Desde el punto de vista de la definición de objetivos, éstos siempre han sido definidos por "el profesor", es decir se ha diseñado y desarrollado para **enseñar**. En un principio el profesor utilizaba un vocabulario no muy claro para transmitir sus necesidades, debido a la falta de experiencia y conocimiento sobre lo que se podía obtener y hacer con un computador; con las distintas experiencias, el profesorado comenzó a conocer los distintos tipos de aplicaciones, los recursos de presentación. Hoy en día, el léxico utilizado para comunicarse y explicar sus requerimientos, se ha enriquecido enormemente, gracias también a demostraciones sobre aplicaciones realizadas en otras latitudes, con distintos conceptos y enfoques.

  Siempre se ha trabajado estrechamente ligado al docente (ambiente de enseñanza) y en muy pocas oportunidades se ha tenido interacción con el estudiante (ambiente de aprendizaje). Sin embargo, se aspira a corto plazo lograr incorporar al estudiante, para conocer el punto de vista y las expectativas del mismo en cuanto a objetivos se refiere.
- En lo que se refiere a los recursos humanos, el equipo de trabajo en general, estaba constituído por el experto en la herramienta a utilizar. Este recibía las solicitudes de desarrollo y con sucesivas reuniones, lograba entender las necesidades de los profesores de los distintos departamentos de la Universidad, los cuales además no tenían un mecanismo formal y concreto para transmitir el esquema y objetivo (a veces difuso) que debía tener la aplicación que se deseaba obtener.

  Con el pasar del tiempo, el profesorado se ha mostrado más interesado en aprender y se le ha educado con la metodología descrita anteriormente, usando como guía las preguntas que se detallan en la fase de diseño de la misma. Nunca ha existido la figura del experto en técnicas de enseñanza, y es "el profesor" quien define los lineamientos que tendrá la aplicación, en base a su experiencia docente y su sentido común. Con la aparición de herramientas fáciles de usar, en los últimos tres años, "el profesor" se ha convertido también en el experto en la herramienta, teniendo a su cargo programadores (estudiantes colaboradores) que se encargan de tareas menos creativas y comprometedoras y más mecánicas como lo son las digitalizaciones de dibujos, generaciones de gráficos, grabaciones de música, voz y efectos especiales, programación de algunas rutinas, etc.

[Canc&Cicc88] CANCIAN, Antonella & CICCARELLI, Lia: "Mycanli: Sistema automatizado para la enseñanza del idioma inglés en la tercera etapa de la educación básica", Escuela de Idiomas, Universidad Metropolitana, Caracas, 1988.

[Carr&Tran89] CARREÑO, Carmen & TRANSMONTE, Lilian; "Aplicación del programa Course Builder™ en el diseño de material de apoyo para el curso Instrumental I de la Universidad Metropolitana", Escuela de Idiomas, Universidad Metropolitana, Caracas 1989.

[Col&Oso&Theo90] COLMENARES, Desiree & OSORIO, Antonella & THEOKTISTO, Víctor; "Fonemas del Castellano hablado en Venezuela", IVAL y Oficina Metrópolis, Universidad Metropolitana, Caracas, 1990.

[Fig&Mill89] FIGARELLA, Elisa &MILLAN, Edna; "Simulación de la Práctica correspondiente al tópico de Filtración", Departamento de Química y Oficina Metrópolis, Universidad Metropolitana, Caracas, 1989.

[Fig&Fin&Mill90] FIGARELLA, Elisa & FINOCCHIO Giancarlo & MILLAN, Edna; "Simulación de la Práctica correspondiente a Ensayos a la Llama", Departamento de Química, Oficina Metrópolis, Universidad Metropolitana, Caracas, 1990.

[Gian86] GIANBERARDINO, Vincenzo; "Vectores", Departamento de Física, Universidad Metropolitana, Caracas, 1986.

[Gian87] GIANBERARDINO, Vincenzo; "Mínimos Cuadrados", Departamento de Física, Universidad Metropolitana, Caracas, 1987.

[Gian90] GIANBERARDINO, Vincenzo; "Simulación de prácticas del laboratorio de Física", Departamento de Física, Universidad Metropolitana, Caracas, 1990.

[Gom&Loe&Lop88] GOMEZ, Ignacio & LOERINS, Peter & LOPATA, Elías; "ODE: Resolución de Ecuaciones Diferenciales Ordinarias", Escuela de Ingeniería de Sistemas, Universidad Metropolitana, Caracas, 1988.

[Hrus87] HRUSKOVEC, Tomislav; "Análisis de Respuesta en Frecuencia", Escuela de Ingeniería Eléctrica, Universidad Metropolitana, Caracas 1987.

[Hrus87] HRUSKOVEC, Tomislav; "Análisis de Flujo de Carga", Escuela de Ingeniería Eléctrica, Universidad Metropolitana, Caracas, 1987.

[Lar&Nuñ89] LARA, Miguel & NUÑEZ, José M.;"Mac370: Emulador de Assembler", Escuela de Ingeniería de Sistemas, Universidad Metropolitana, Caracas, 1989.

[Mene&Rami88] MENEGHINI, Roberto & RAMIREZ, Rodrigo; "Manejador de Archivos Indexados VSAM", Escuela de Ingeniería de Sistemas, Universidad Metropolitana, Caracas, 1988.

[Mill88] MILLAN, Edna; "Teorema del Valor Medio", Oficina Metrópolis, Universidad Metropolitana, Caracas, 1988.

[Rami87] RAMIREZ, Rodrigo; "FuncLib: Libreria de Rutinas Matemáticas", Escuela de Ingeniería de Sistemas, Universidad Metropolitana, Caracas, 1987.

[Rami&Theo87] RAMIREZ, Rodrigo & THEOKTISTO, Víctor; "Tabla Periódica", Oficina Metrópolis, Universidad Metropolitana, Caracas, 1987.

[Rint87] RINTELEN, Alfonso; "PerfoPlan", Escuela de Ingeniería Civil e Ingeniería Eléctrica, Universidad Metropolitana, Caracas, 1987.

[Stre88] STREIT, Siegfried; "MacEduca", Escuela de Ingeniería Eléctrica, Universidad Metropolitana, Caracas, 1988.

[Theo87] THEOKTISTO, Víctor; "Enlaces Químicos", Oficina Metrópolis, Universidad Metropolitana, Caracas,1987.

A neural network based approach to sensor fault detection

Abhay Bulsari†, Alexander Medvedev* and Henrik Saxén

Kemisk-tekniska fakulteten, Åbo Akademi
SF 20500 Turku/Abo, Finland.
Tel: 358 (21) 654 311
Fax: 358 (21) 654 479
E-mail: vt_ai@abo.fi

ABSTRACT

This work investigated the a new approach, combining two recently developed techniques to detect sensor faults in dynamic systems, exemplified by an application to an industrially vital biochemical process.

State vector estimation, the first part of the sensor fault detection procedure, is based on delayed measurements of the control and output vectors. It assumes linear dynamics and incorporates a priori information regarding system dynamics. Observer residuals are calculated as the difference between the two estimations of the plant state vector, one excluding one or more last measurements, and the other based on the all the output measurements available for the estimator at each moment. These residuals are fed to a feed-forward neural network, which is trained to recognise which particular sensor(s) is/are faulty.

Levenberg-Marquardt method was used to train the networks by error square sum minimization. The method is known to be fast, accurate and reliable. The output of each node was calculated using the logistic (sigmoid) activation function on the weighted sum of inputs to that node. The networks are trained for single as well as multiple faults in sensors. The biases were very small, indicating a good choice of input variables and training instances.

The linearised third order dynamic model was used for the state vector estimator. The neural networks had three inputs since there were three state variable estimates, and had three outputs used as fault indicators for each of the three sensors. The most appropriate choice for the network configuration was found to be (3,17,3) based on the training experience with 707 training instances.

The trained neural networks successfully identified single and multiple faults during the testing phase, when they were fed with data which they were not trained on. This work demonstrated that the technique works reliably for the problem considered in this paper. This technique can be implemented in real time systems since the computing power required is quite low, once the network is trained.

---

† on leave from Lappeenranta University of Technology
* on leave from Leningrad Electrical Engineering Institute

1. Introduction

Failures in a system are detectable if the outputs following the failure are different from the outputs prior to the failure. Therefore, the process history should be compared with the current plant state to detect possible system malfunctions. Such comparison can be performed by using delayed-data driven observer which computes a number of estimates – one based on process history and the other based on process history and more recent measurement data.

If the recent measurements do not fit the values predicted from process history, then they are most likely provided by faulty sensor readings.

Another objective to be achieved is fault isolation, *i.e.* a procedure of differentiating between failures which may occur in a system. Fault isolation procedure requires a substantial amount of logical operations due to decision making process involved. This task can be successfully completed by artificial neural networks.

Artificial neural networks (ANNs) offer interesting possibilities in chemical and biochemical engineering. Various configurations and topologies of ANNs have been suggested [1] . ANNs, also called connectionist models, parallel distributed models, neuromorphic systems or simply neural nets consist of many simple computational elements called nodes, or neurons each of which collects by weighted addition the signals from various other nodes which are connected to it directionally. This sum, the net input to the node, is processed by a function (usually a sigmoid or a step) resulting in the output or the activation of that node. In multilayer feed–forward networks of the type considered in this paper (Fig. 1), there are a few layers (the input layer, the output layer, and possibly some hidden layers) across which all the nodes of each layer are connected to all the nodes in the layer above it, but there are no connections within the layer. A constant term, a bias, is usually added to each

of the nodes.

Although back propagation has become popular on grounds of simplicity and its capability to learn sequentially from training instances, we have used the Levenberg–Marquardt method [2, 3] for minimising the sum of squares of errors. Levenberg–Marquardt method is fast, reliable and easy to use when the number of weights is not very large, say less than 100. It has been successfully used for a system identification task without dynamics [4] .

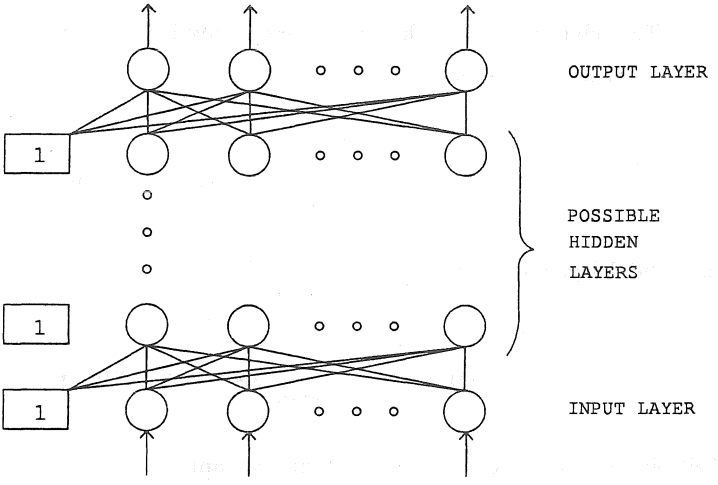


Figure 1. Feed–forward neural network

2. Deterministic Least Squares Estimator

In this section, we address the problem of linear dynamic system state

vector reconstruction in presence of sensor failures. Let us consider the nonhomogeneous equation

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y_i \triangleq y(t-\tau_i) &= C_i x(t-\tau_i), \qquad i = 0, 1, \ldots, k \end{aligned} \tag{2.1}$$

where $x(t) \in R^n$ is the state vector, $u(t) \in R^m$ is the control vector, $y(t) \in R^l$ is the observation vector, and y_i are delayed observation vector values. A, B, C_i are real matrices of appropriate dimensions.

The relation between the current and delayed state vector values is given by the state-space equation

$$x(t) = \exp(A\tau_i)x(t-\tau_i) + \int_{t-\tau_i}^{t} \exp(A(t-\vartheta))Bu(\vartheta)d\vartheta \tag{2.2}$$

Then the delayed state vector $x(t-\tau_i)$ is as follows

$$x(t-\tau_i) = \exp(-A\tau_i)x(t) - \int_{t-\tau_i}^{t} \exp(A(t-\tau_i-\vartheta))Bu(\vartheta)d\vartheta \tag{2.3}$$

Multiplying (2.3) by C_i from the left and rearranging gives

$$C_i \exp(-A\tau_i)x(t) = y_i + \int_{t-\tau_i}^{t} C_i \exp(A(t-\tau_i-\vartheta))Bu(\vartheta)d\vartheta$$

If the linear system (1.1) output is measured $k+1$ times, then the state-vector could be found through solution of linear equation

$$W_k x(t) = Y_k \tag{2.4}$$

where $W_k \in R^{(k+1)l \times n}, Y_k \in R^{(k+1)l \times 1}$ are real matrices

$$W_k = \begin{pmatrix} C_0 \exp(-A\tau_0) \\ \cdot \\ \cdot \\ \cdot \\ C_k \exp(-A\tau_k) \end{pmatrix},$$

$$Y_k = \begin{pmatrix} y(t-\tau_0) + \int_{t-\tau_0}^{t} C_0 \exp\left(A(t-\vartheta-\tau_0)\right) B u(\vartheta) d\vartheta \\ \cdot \\ \cdot \\ \cdot \\ y(t-\tau_k) + \int_{t-\tau_k}^{t} C_k \exp\left(A(t-\vartheta-\tau_k)\right) B u(\vartheta) d\vartheta \end{pmatrix}$$

It is possible to solve equation (2.4) in the least-squares sense so that

$$(Y_k - W_k x)^T R^{-1} (Y_k - W_k x)$$

is minimized. The R is arbitrary with $R^{-1} = Q^T Q$ a positive-definite weighting matrix. The least-squares solution $\hat{x}_k$ is given by

$$\hat{x}_k(t) = (QW_k)^+ QY \tag{2.5}$$

where $(\cdot)^+$ denotes Moore-Penrose pseudoinverse of a matrix.

If $\mathrm{rank}(W_k) = n$ and R is unit matrix then from (2.4),

$$\hat{x}_k(t) = (W_k^T W_k)^{-1} W_k^T Y \tag{2.6}$$

The more natural form of the existence condition for analytical solution of (2.4) is the positive definiteness of the square real matrix $W_k^T W_k$;

$$\mathrm{rank}(W_k) = \mathrm{rank}(W_k^T W_k) = \mathrm{rank}\left(\sum_{i=0}^{k} (\exp(-A^T \tau_i) C_i^T C_i \exp(-A\tau_i))\right)$$

or, in terms of nonsingularity

$$\det\left(\sum_{i=0}^{k} (\exp(-A^T \tau_i) C_i^T C_i \exp(-A\tau_i))\right) \neq 0 \tag{2.7}$$

After multiplication of block matrices in (2.6) the estimate computed from $k+1$ sample output values $i = 0, \ldots, k$ can be rewritten in the form

$$\hat{x}_k(t) = \mathcal{W}_k^{-1} \sum_{i=0}^{k} \exp(-A^T \tau_i) C_i^T (y(t-\tau_i) + \int_{t-\tau_i}^{t} C_i \exp(A(t-\delta-\tau_i)) B u(\delta) d\delta) \qquad (2.8)$$

where

$$\mathcal{W}_k = W_k^T W_k = \sum_{i=0}^{k} \exp(-A^T \tau_i) C_i^T C_i \exp(-A\tau_i)$$

3. Fault Detection Window

Let us consider the application of the observation algorithm (2.8) to sensor fault detection, using time-redundancy principle. The main idea is the comparison of two estimates: one based on process output history, excluding r last measurements, and the other based on the all k output measurements available for the estimator at each moment. The r value will be refered as fault detection window (which lasts actually from $t - \tau_{k-r}$ to t), because it contains data suspected to be provided by a faulty sensor.

We assume that there are no sensor failures before the moment $t - \tau_{k-r}$. Our purpose is to detect a fault and a faulty sensor inside the fault detection window by means of algorithm (2.8).

Define $\hat{x}_k(t)$ as the estimate computed from $k+1$ sample output values $i = 0, \ldots, k$ and $\hat{x}_{k-r}(t)$ the estimate computed from $k-r$ sample output values $i = 1, \ldots, k-r$. Taking into account matrices partition, one may rewrite (2.4) as

$$\begin{pmatrix} W_{k-r} \\ W_r \end{pmatrix} x(t) = \begin{pmatrix} Y_{k-r} \\ Y_r \end{pmatrix} \qquad (3.1)$$

Using partitioned matrices notation for the corresponding estimates, we get

$$\hat{x}_{k-r} = W_{k-r}^{+} Y_{k-r}$$

$$\hat{x}_k = W_k^{+} Y_k$$

The well-known formula for the partitioned matrix pseudoinverse (Boullion and Odell, 1971) [10] gives the estimate in terms of submatrices

$$\hat{x}_k = (W_{k-r}^{+} - T_r W_r W_{k-r}^{+},\ T_r)\, Y_k$$

where

$$T_r = F_r^{+} + (E - F_r^{+} W_r) W_{k-r}^{+} {W_{k-r}^{+}}^{T} W_r^{T} K_r (E - F_r F_r^{+})$$

$$K_r = \left(E + (E - F_r F_r^{+}) W_r {W_{k-r}^{+}} {W_{k-r}^{+}}^{T} W_r^{+} (E - F_r F_r^{+})\right)^{-1}$$

$$F_r = W_r (E - W_{k-r}^{+} W_{k-r})$$

Eventually, the least-square solution for (3.1) is

$$\hat{x}_k = (W_{k-r}^{+} - T_r W_r W_{k-r}^{+}) Y_{k-r} + T_r Y_r \tag{3.2}$$

The whole information about possible sensor fault contains in the difference between the reliable estimate $\hat{x}_{k-r}$ and the estimate $\hat{x}_k$ which includes also unreliable data

$$\varepsilon_r = \hat{x}_{k-r} - \hat{x}_k = T_r (W_r W_{k-r}^{+} Y_{k-r} - Y_r) = T_r (W_r \hat{x}_{k-r} - Y_r) \tag{3.3}$$

Now it is easy to see that the ε_r vector demonstrates how fault detection window data fits process history. If $\mathrm{rank}(W_{k-r}) = n$ then the equation for T_r can be considerably simplified

$$F_r = 0$$

$$K_r = \left(E + W_r W_{k-r}^{+} {W_{k-r}^{+}}^{T} W_r^{+}\right)^{-1}$$

$$T_r = W_{k-r}^{+} {W_{k-r}^{+}}^{T} W_r^{T} K_r$$

Moreover, pseudoinverse operation for the W_{k-r} matrix can be substituted by $W_{k-r}^{+} = (W_{k-r}^{T} W_{k-r})^{-1} W_{k-r}^{T}$. Now it follows that

$$W_{k-r}^{+} {W_{k-r}^{+}}^{T} = (W_{k-r}^{T} W_{k-r})^{-1} = \mathcal{W}_{k-r}^{-1}$$

and finally the difference between estimates for the fault detection window width r is

$$\begin{aligned} \varepsilon_r &= T_r(W_r\hat{x}_{k-r} - Y_r) \\ T_r &= W_{k-r}^{-1}W_r^T(E + W_rW_{k-r}^{-1}W_r^T)^{-1} \end{aligned} \tag{3.4}$$

Then, if all sensors are operating properly the estimate $\hat{x}_{k-r}$ and the estimate $\hat{x}_k$ will be nearly identical. If, however, a sensor fault occurs at the moment inside the fault detection window, the ε_r value would be of sufficient scale when the output change caused by this sensor fault is noticeable. To provide more information for decision making under fault detection/isolation procedure a number of ε_i can be generated. Since all matrix coefficients $T_i, i = 1,\ldots,r$ are independent from the current measurements and do not use any additional information in comparison with the estimate $\hat{x}_k$, the set of $\varepsilon_i, i = 1,\ldots,r$ seems to be reasonable choice.

4. The feed–forward neural network

The feed–forward neural network has an input layer which simply transmits the input variables without any processing to the next layer. The bias is a weighted unit input to each node, thus adding a constant term to the net input of the node. Two, one or no hidden layers are considered. Each node in the upper layers (hidden or output layers) receives weighted inputs from each of the nodes in the layer below it. These weighted inputs are added to get the net input of the node, and the output x_i (or the activation) is calculated by an activation function of the net input a_i as

$$a_i = \sum_{j=0}^{N} w_{ij}x_j \tag{4.1}$$

$$x_i = \frac{1}{1 + e^{-\beta a_i}} \tag{4.2}$$

where N is the number of nodes in a hidden layer or the number of input nodes and w_{i0} is the bias of node i. β is referred to as the gain term and is usually set to 1.

5. The biochemical process

Biochemical processes have highly non–linear characteristics, and have operability in limited domains. In the process considered here, *Saccharomyces cerevisiae*, a yeast, is grown on glucose substrate in a chemostat (a biochemical continuous stirred tank reactor) producing ethanol as a product of primary energy metabolism. There are three state variables : microbial concentration, X; substrate concentration, S; and product concentration, P. The kinetic and stoichiometric parameters were taken from a recent study on kinetics of this system [5] . The feed to the chemostat is sterile, *i.e.* there are no microorganisms in the feed. The feed concentration of substrate, glucose, is S_0, and D is the dilution rate (volumetric flow rate per volume of the chemostat.) See Fig. 2. The dynamics of this system can be described by the following equations.

$$\frac{dX}{dt} = (\mu - D)X \tag{5.1}$$

$$\frac{dS}{dt} = D(S_0 - S) - Y_{S/X}\mu X \tag{5.2}$$

$$\frac{dP}{dt} = -DP + Y_{P/X}\mu X \tag{5.3}$$

where the growth rate, μ and the yield coefficients, $Y_{S/X}$ and $Y_{P/X}$ are given by

$$\mu = \frac{0.427S}{0.245 + S}(1 - (P/101.6)^{1.95}) \tag{5.4}$$

$$Y_{P/X} = 3.436, \qquad Y_{X/P} = 0.291 \tag{5.5}$$

$$Y_{X/S} = 0.152(1 - P/302.3), \qquad Y_{S/X} = 1/Y_{X/P} \tag{5.6}$$

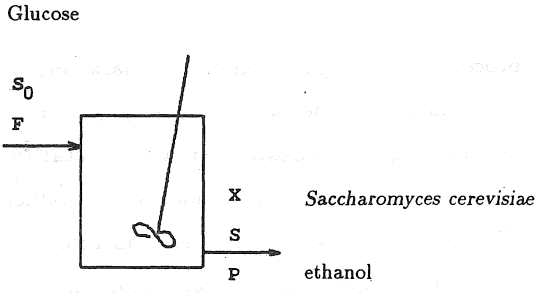


Figure 2. A bioreactor

The growth rate, μ cannot exceed 0.427, and that is also the upper limit for the dilution rate, D. If the dilution rate is higher than the growth rate, microbial concentration in the bioreactor decreases to zero. Table 1 shows typical steady state values for the process parameters.

5.1. Linearised equations of the process

This can be seen from the linearised set of equations at the operating parameters given in Table 1. Variables in small letters are the deviation from their corresponding operating parameters.

Table 1. Typical steady state values of the process parameters.

S_0	101.416 gm/lit
D	0.345 hr^{-1}
X	12.0 gm/lit
S	10.0 gm/lit
P	41.232 gm/lit
μ	0.345 hr^{-1}
$Y_{X/S}$	0.1313
$Y_{X/P}$	0.291

$$\begin{pmatrix} dx/dt \\ ds/dt \\ dp/dt \end{pmatrix} = \begin{pmatrix} 0 & -0.04075 & 0.00990 \\ 1.18542 & -0.48503 & 0.03402 \\ -2.62821 & 0.189656 & -0.42042 \end{pmatrix} \begin{pmatrix} x \\ s \\ p \end{pmatrix} + \begin{pmatrix} 0 & -12. \\ 0 & -41.232 \\ 0.345 & 91.416 \end{pmatrix} \begin{pmatrix} s_0 \\ d \end{pmatrix}$$

6. The Levenberg–Marquardt method

Levenberg–Marquardt method [2, 3] was used to determine the weights in the neural networks. Network training aims at minimising the sum of squares of errors, the errors measured as the difference between the calculated output and the desired output. Minimising the SSQ is not always the best way of training a neural network as pointed out in [4] , but for this application, it suffices. Back propagation by the generalised delta rule, a kind of a gradient descent method is one popular method [8] for training feed–forward neural networks. Many algorithms for least–squares optimization use Taylor–series models. The Levenberg–Marquardt method uses an interpolation between the

approaches based on the maximum neighbourhood (a "trust region") in which the truncated Taylor series gives an adequate representation of the non–linear model. The method has been found to be quite efficient.

7. Training the neural networks

Neural networks were first trained for detecting just one fault. The training set consisted of 404 training instances. Each training instance includes three observer residuals followed by 1 for a faulty sensor, and 0 for a sensor in working condition. Initially, the whole state vector is supposed to be measureable. At the moment of a simulated sensor fault, a drastic change takes place in all residuals, which are the subject of the fault detection / isolation procedure performed by the neural network. An example of the observer residuals is shown in Fig. 3. The window width was $r = 1$ throughout this work.

The error square sum (SSQ) was 13 for (3,3), and for one hidden layered networks, it reduced to 12 when the number of hidden nodes was increased to (3,4,3). (3,8,3) and (3,9,3) resulted in a SSQ of 4, and (3,10,3) resulted in 0.01495. Two hidden layered networks were also considered, and the SSQ for (3,3,3,3) was 4. All these were trained with a β of 100. While most of the one hidden layered networks were trained using the program ANNEX [9] (with the Levenberg–Marquardt method), two hidden layered networks were often trained by simulated annealing, until the weights were close enough to the minimum, after which they were further improved by ANNEX.

The (3,10,3) was then tested for single faults on a set of data it was not trained on. It worked correctly for all the cases therein. It, however, failed to detect mulitple sensor faults, as was expected.

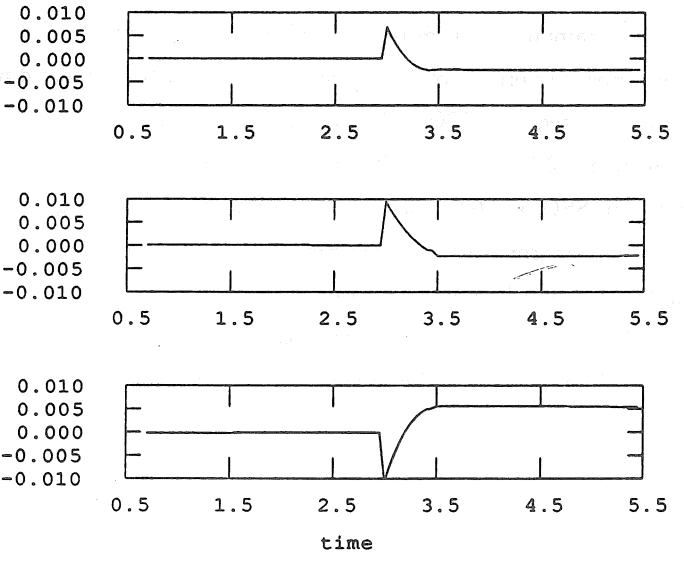


Figure 3. Observer residuals for the three state variables.
Fault in sensors 2 and 3 occurring at t = 3.00

Therefore, neural networks were then trained for detecting just two faults. The 303 training instances consisted of faults in sensors 1 and 2, faults in sensors 2 and 3, faults in sensors 1 and 3, and no faults. The network without hidden layers (3,3) had a SSQ of 111.37. (3,5,3) resulted in a SSQ of 11.692, (3,8,3) in 4.035, and (3,11,3) in 1.000. The SSQ for (3,12,3) was 0.188×10^{-10}. This trained network correctly identified the faulty sensors after a small time lag, when the test cases had two faults, but failed to identify the faulty sensor when only one sensor was faulty. It was also observed that (3,11,3) performed better than (3,12,3). This kind of over-fitting of data is known to happen with feed-forward neural networks.

Finally, the training instances from the above two sets were put together in a training set, now comprising of 707 instances. This was fairly more difficult to train. (3,9,3) resulted in a SSQ of 51.41, and (3,10,3) in 24.56. There was little improvement in (3,14,3) with a SSQ of 24.40. However, the (3,15,3) and (3,16,3) resulted in SSQ of 6.000. (3,17,3) was found to have a SSQ of 2.000. This network could recognise 0, 1, or 2 correct faults in sensors. There is time lag of 10 sampling intervals before it can correctly point the faulty sensors, which is understandable because the algorithm for calculating ϵ requires 10 samples of measurements. More details of the testing are given in the next section.

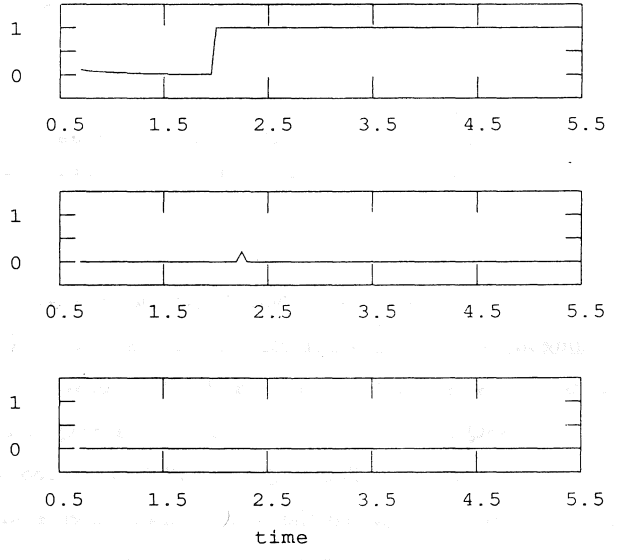


Figure 4. Detected faults in sensors 1, 2, 3 using the trained network (3,10,3)

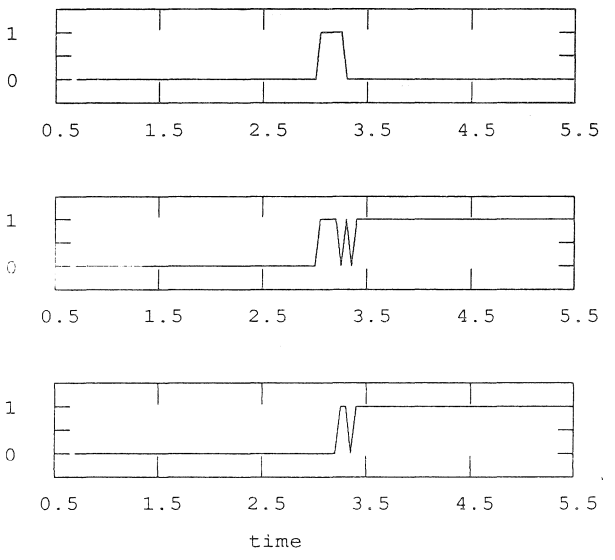


Figure 5. Detected faults in sensors 1, 2, 3
using the trained network (3,12,3)

8. Testing the neural networks

Networks trained for the detection of one faulty sensor could correctly detect the faulty sensor, when there was only one faulty sensor. Fig. 4 shows the results of the (3,10,3) neural network's output when one sensor (the first one) is faulty after $t = 2.00$ However, it could not correctly detect faults in two sensors simultaneously. Networks trained for detecting two faulty sensors could do it (see Fig. 5), but they could not detect the faulty sensor, when only one was faulty.

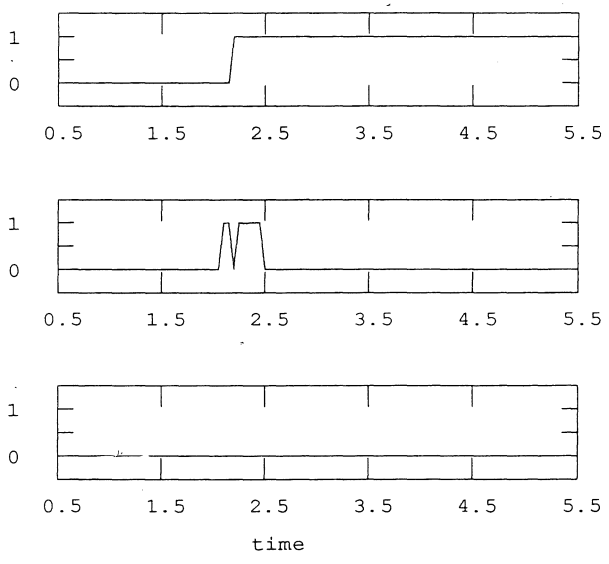


Figure 6. Detected faults in sensors 1, 2, 3 using the trained network (3,17,3)

The (3,17,3) network trained on 707 instances with 0, 1 or 2 faults could detect 0, 1 or 2 faults correctly after a time lag of about 10 sampling intervals. This is shown in Figs. 6 and 7. Fig. 6 shows the results of the neural network's output when one sensor (the first one) is faulty after $t = 2.00$, and each sampling interval is 0.05. Thus, the fault is recognised correctly after 10 sampling intervals. Similarly, Fig. 7 shows the results of the neural network's output when two sensors (2 and 3) are faulty after $t = 3.00$. Note that in both cases, the neural network output is consistently 0 when no fault is present.

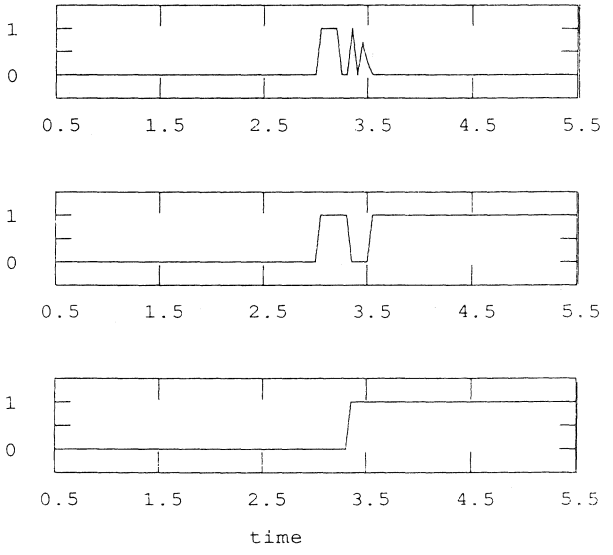


Figure 7. Detected faults in sensors 1, 2, 3 using the trained network (3,17,3)

9. Results and conclusions

The combination of using state vector estimation with neural networks proved to be effective. The estimator provides the neural network with substantial information on the sensor fault through the system dynamic model and a process history. It was shown that observer residuals are unique for each fault type. The neural network uses this information for fault detection and isolation. As shown by simulations, the proposed method successfully detected single and mulitple sensor faults in a linearised dynamic system.

For the third order biochemical system considered in this work, the network sizes required for detecting one faulty sensor, and one or two faulty sensors at a time were (3,10,3) and (3,12,3) respectively. The (3,10,3) network trained for one fault could not isolate two simultaneous faults, and the (3,12,3) trained for two faults coudl not reliably isolate a single fault. (3,17,3) was trained for, and could recognise 0, 1, or 2 faults. Neural networks were trained using the Levenberg–Marquardt method, which was once again found to be a good choice compared to the conventional back-propagation or its variants. Training is usually not so straightforward since there is always a possibility of getting into a local minimum. In most cases, very sharp sigmoids were used (with a gain of 100, looking almost like hard–limiter step functions), which would have made it very difficult for back–propagation–like methods to converge to reasonable solutions. Faults were detected almost at the point of occurrence, but there was usually some time lag before the fault(s) were isolated. This time lag was 10 delayed measurements or less, the amount required by the estimator.

References

1. Lippmann, R. P.,
"An introduction to computing with neural nets",
IEEE ASSP Magazine, (April 1987) 4-22.

2. Marquardt, D. W.,
"An algorithm for least-squares estimation of nonlinear parameters",
J. Soc. Indust. Appl. Math., 11 (June 1963) 431-441.

3. Levenberg, K.,
"A method for the solution of certain nonlinear problems in least squares",
Quart. Appl. Math., 2 (1944) 164-168.

4. Bulsari, A. and H. Saxén,
"Applicability of an artificial neural network as a simulator for a chemical process",
Proceedings of the fifth International Symposium on Computer and
Information Sciences, Nevsehir, Turkey (October 1990) 143-151.

5. Warren, R. K., G. A. Hill and D. G. Macdonald,
"Improved bioreaction kinetics for the simulation of continuous ethanol
fermentation by *Saccharomyces cerevisiae*",
Biotechnology Progress, 6 (1990) 319-325.

6. Medvedev, A.,
"Three approaches toward a fault tolerant controller structure",
Technical report 91–3, Reglerteknik, Åbo Akademi (1991)

7. Silverman, L. M. and M. Bettayeb,
"Optimal approximation of linear systems",
Proceedings of the Joint American Control Conference, (1980)

8. Jones, W. P. and J. Hoskins,
"Back-Propagation: A generalized delta learning rule"
Byte, (October 1987) 155-162.

9. Bulsari, A., B. Saxén and H. Saxén,
"Programs for feedforward neural networks using the Levenberg-Marquardt
method : Documentation and user's manual",
Technical report 90-2, Värmeteknik, Åbo Akademi, Finland (1990).

10. Bouillon, T. L. and P. L. Odell,
"Generalised inverse matrices",
Wiley Interscience, New York, (1971)

Un Circuito de Redes Neurales para la Interpretación de Patrones Ambiguos Mediante la Extracción de Autovectores

Leopoldo Mauro, EXPERT Computación C.A., Caracas
Andreas Meier, Universidad Simón Bolívar, Caracas

RESÚMEN

La característica principal de las redes neurales es su capacidad de reconocer y aprovechar regularidades en los datos, usándolas para clasificar patrones desconocidos. La mayoría de las redes neurales son capaces de dar sólo una clasificación de cada patrón, no un conjunto de clasificaciones alternas, y tampoco indican la confiabilidad de sus resultados. Además, no pueden reconocer nuevos patrones compuestos por sub-patrones conocidos, a menos que se las entrene antes con todas las combinaciones posibles de sub-patrones.

Fundamentándonos en los métodos de extracción iterativa de autovectores empleados en álgebra lineal, proponemos un circuito de tres redes neurales autónomas, que juntas exhiben la capacidad de producir todas las interpretaciones de un patrón en orden decreciente de importancia, descomponiéndolo en sub-patrones conocidos si es posible.

Describimos además un ejemplo de diseño y ensamblaje de redes neurales a partir de componentes modulares más simples, similar a la metodología empleada en electrónica. Esta clase de circuitos tendrán un papel determinante, sobre todo cuando finalmente aparezcan en el mercado los circuitos integrados neurales.

Un Circuito de Redes Neurales para la Interpretación de Patrones Ambiguos Mediante la Extracción de Autovectores

Leopoldo Mauro, EXPERT Computación C.A., Caracas
Andreas Meier, Universidad Simón Bolívar, Caracas

1. Introducción.

La capacidad principal de la técnica de redes neurales consiste en reconocer, almacenar y aprovechar regularidades en un conjunto de datos. La redes neurales pueden estructurar espacios de datos de manera autónoma, y reconocen con gran facilidad la pertenencia de datos de prueba a alguna de esas estructuras; es decir, reconocen y clasifican patrones.

La característica dominante de las técnicas de clasificación mediante redes neurales es que cada patrón de entrada produce un sólo resultado, y no un conjunto de posibles interpretaciones alternas. Generalmente, una red neural tampoco indica la confiabilidad de sus resultados, por lo que un patron de entrada ambiguo es clasificado según alguna de sus posibles interpretaciones sin indicación del hecho. Más aún, la mayoría de las redes no pueden ser entrenadas con patrones ambiguos sin incurrir en alguna penalidad, a veces severa, en la capacidad de almacenamiento o en la confiabilidad de la clasificación. Además, la mayor parte de las redes neurales no pueden detectar que un patrón desconocido está compuesto por dos o más patrones conocidos, a no ser que hayan sido entrenadas previamente con todas las posibles combinaciones de patrones. Esta última es una grave limitación de muchas de las redes neurales existentes, que no parece ser compartida por nuestro cerebro.

El presente trabajo propone un circuito de tres redes neurales autónomas que colaboran para dar al conjunto la capacidad de producir todas las posibles interpretaciones de un patrón en orden decreciente de importancia, y descomponer patrones desconocidos en sus componentes conocidos. Adicionalmente, el circuito produce un "vector de novedad" que contiene los componentes del patrón de entrada que no pudieron interpretarse.

La métodología que proponemos consiste en diseñar y ensamblar redes neurales heterogéneas a partir de componentes modulares más simples de propiedades conocidas, en forma muy similar a lo que se hace en electrónica. La literatura de redes neurales habla muy poco al respecto, con una importante excepción (Fukushima 1980, Fukushima et al. 1983).

2. Métodos de Extracción de Autovectores.

Toda matriz cuadrada ($\mathcal{N}\times\mathcal{N}$) posee $\mathcal{N}$ autovectores y autovalores (algunos de los cuales pueden ser iguales); si además es una matriz real simétrica, sus autovectores y autovalores serán todos reales y ortogonales entre sí. Esta última propiedad es muy importante, pues las matrices de conexión de las redes neurales autoasociativas que se discutirán en este trabajo son todas reales y simétricas.

La extracción algorítmica de autovectores es un tema muy conocido en el álgebra lineal (un buen texto introductorio es Jennings 1977). En el análisis tradicional se ha hecho bastante énfasis en los métodos de extracción directa por transformación de la matriz (diagonalización de Jacobi, transformación de Householder, transformación QR, etc.) por sus ventajas en lo referente a tiempo de cómputo, requerimientos de memoria y errores de redondeo. Pero desde el punto de vista de la computación neural, los métodos iterativos (*power method* y purificación de Aitken) son superiores pues pueden realizarse directamente en el paradigma neural.

2.1. Power Method.

El fundamento del *power method* es el siguiente. Sea $\mathbf{u}$ un vector cualquiera, èxpresado en la base de los autovectores de una matriz simétrica $\mathbf{W}$ como

$$\mathbf{u} = \rho_1 \mathbf{q}_1 + \rho_2 \mathbf{q}_2 + \ldots + \rho_n \mathbf{q}_n \tag{1}$$

Obtengamos

$$\begin{aligned}\mathbf{u}^{(1)} = \mathbf{W}\,\mathbf{u} &= \mathbf{W} \sum_i \rho_i \mathbf{q}_i = \sum_i \rho_i \mathbf{W}\,\mathbf{q}_i \\ &= \rho_1\lambda_1\mathbf{q}_1 + \rho_2\lambda_2\mathbf{q}_2 + \ldots + \rho_n\lambda_n\mathbf{q}_n \end{aligned} \tag{2}$$

Repitiendo esta operación k veces se obtiene

$$\mathbf{u}^{(k)} = \mathbf{W}\,\mathbf{u}^{(k-1)} = \rho_1 \lambda_1^k \mathbf{q}_1 + \rho_2 \lambda_2^k \mathbf{q}_2 + \ldots + \rho_n \lambda_n^k \mathbf{q}_n \tag{3}$$

Si hemos numerado los autovectores de tal manera que $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$, podemos hacer la aproximación

$$\mathbf{u}^{(k)} \approx \rho_1 \lambda_1^k \mathbf{q}_1 \tag{4}$$

para k suficientemente grande. Vemos que $\mathbf{u}^{(k)}$ tiende a ser proporcional a $\mathbf{q}_1$, el autovector dominante de $\mathbf{W}$, a medida que k crece.

Puesto que los autovectores no cambian ante un escalamiento, resulta conveniente evitar el crecimiento indefinido de $\mathbf{u}^{(k)}$ siguiendo a cada premultiplicación por $\mathbf{W}$ con una normalización. El *power method* puede entonces resumirse en las siguientes ecuaciones recurrentes

$$\mathbf{v}^{(k)} = \mathbf{W}\,\mathbf{u}^{(k)} \tag{5a}$$

$$\mathbf{u}^{(k+1)} = \phi\left(\mathbf{v}^{(k)}\right) \tag{5b}$$

donde ϕ es una función vectorial de normalización, usualmente una de las siguientes

$$\phi(\mathbf{x}) = \mathbf{x} \,/\, \|\mathbf{x}\| \tag{6}$$

$$\phi(\mathbf{x}) = \mathbf{x} \,/\, \max x_i \tag{7}$$

El *power method* siempre puede aplicarse a matrices simétricas. Su convergencia es buena si los autovectores dominantes poseen autovalores diferentes, pero en el caso de autovalores similares la convergencia puede ser muy lenta.

2.2. Purificación de Aitken.

Si el *power method* permite obtener el autovector dominante de una matriz, el método conocido como "purificación de Aitken" (Aitken 1937) permite obtener los demás autovectores, o autovectores subdominantes. Este método está fundamentado en que, si la matriz simétrica $\mathbf{W}$ posee a $\mathbf{q}_1$ como autovector de autovalor λ_1, la nueva matriz

$$\mathbf{W}' = \mathbf{W} - \left(\lambda_1 / \|\mathbf{q}_1\|^2\right) \mathbf{q}_1 \mathbf{q}_1^T \tag{8}$$

posee los mismos autovectores que $\mathbf{W}$, con la excepción que el autovalor asociado con $\mathbf{q}_1$ en $\mathbf{W}'$ es 0. Esto es fácil de demostrar premultiplicando $\mathbf{q}_1$ por $\mathbf{W}'$:

$$\begin{aligned} \mathbf{W}'\,\mathbf{q}_1 &= \mathbf{W}\,\mathbf{q}_1 - \left(\lambda_1 / \|\mathbf{q}_1\|^2\right) \mathbf{q}_1 \mathbf{q}_1^T \mathbf{q}_1 \\ &= \lambda_1 \mathbf{q}_1 - \left(\lambda_1 / \|\mathbf{q}_1\|^2\right) \mathbf{q}_1 \|\mathbf{q}_1\|^2 \\ &= (\lambda_1 - \lambda_1)\,\mathbf{q}_1 = 0\,\mathbf{q}_1 \end{aligned} \tag{9}$$

Este proceso puede extenderse a la eliminación de varios autovectores repitiendo las operaciones indicadas en (8) para cada uno. El método de Aitken para la obtención de autovectores subdominantes consiste, entonces, en la obtención del autovector dominante por medio del *power method*, y su eliminación por medio de (8). Al aplicar nuevamente el *power method* ya no se obtendrá más $\mathbf{q}_1$, sino el siguiente autovector en orden decreciente de autovalor, el cual puede nuevamente eliminarse repitiendo el proceso de purificación.

Esta metodología de purificaciones sucesivas es sensible a los errores en la determinación del autovector a ser eliminado, ya que la iteración del *power method* amplificará dichos errores hasta que interfieran con la convergencia. Por ello el método de Aitken rara vez se emplea para obtener más que los primeros pocos autovectores de una matriz.

3. Redes Neurales Autoasociativas.

En términos generales, una red neural autoasociativa es aquella que, por diseño o entrenamiento, es capaz de reproducir en su salida una copia (posiblemente "limpia") de su entrada. En este artículo nos centraremos tan sólo en las redes autoasociativas de una sóla capa con arquitectura recurrente. Estas redes se caracterizan por poseer una matriz de conexión simétrica, cuyos autovectores dominan la evolución en el tiempo del sistema iterativo resultante. En efecto, el propósito de estas redes es el de reproducir en su salida al vector de entrada, con tal que éste se encuentre entre aquellos que la red reconoce (sus autovectores).

Cuando el vector de prueba no corresponde exactamente a algún vector reconocido, la red ha de responder con el vector conocido más cercano (de acuerdo a cierta métrica). En otras palabras, el vector de prueba es descompuesto de acuerdo a la base definida por los autovectores de la matriz de conexión, y el componente de mayor norma (o más importante) es el resultado de la red.

3.1. Matrices Autoasociativas.

Las redes neurales autoasociativas recurrentes se basan en la siguiente observación (Amari 1977, Anderson et al. 1977). Tomemos una matriz autoasociativa $\mathbf{W}$ construida a partir de un vector columna $\mathbf{a}$ según la siguiente operación (reminiscente del método de Aitken):

$$\mathbf{W} = \mathbf{a}\,\mathbf{a}^T \tag{10}$$

Multiplicando dicha matriz por el "vector de prueba" $\mathbf{p}=\mathbf{a}$ obtenemos

$$\mathbf{a}' = \mathbf{W}\,\mathbf{p} = \mathbf{a}\,\mathbf{a}^T\,\mathbf{a} = \mathbf{a}\,\|\mathbf{a}\|^2 = \lambda\,\mathbf{a} \tag{11}$$

En otras palabras, la matriz $\mathbf{W}$ posee al vector $\mathbf{a}$ como uno de sus autovectores, con autovalor $\lambda=\|\mathbf{a}\|^2$ (positivo para todo $\mathbf{a}$ no nulo).

Es posible construir la matriz $\mathbf{W}$ combinando múltiples vectores $\mathbf{a}_i$ (usualmente denominados "vectores de entrenamiento"):

$$\mathbf{W} = \sum_i \mathbf{a}_i\,\mathbf{a}_i^T \tag{12}$$

Al multiplicar $\mathbf{W}$ por un vector de prueba $\mathbf{p}=\mathbf{a}_j$ tendremos

$$\begin{aligned}\mathbf{a}_j' = \mathbf{W}\,\mathbf{p} &= \Big(\sum_i \mathbf{a}_i\,\mathbf{a}_i^T\Big)\,\mathbf{a}_j \\ &= \mathbf{a}_j\,\mathbf{a}_j^T\,\mathbf{a}_j + \sum_{i\neq j} \mathbf{a}_i\,\mathbf{a}_i^T\,\mathbf{a}_j \\ &= \lambda_j\,\mathbf{a}_j + \sum_{i\neq j} \mathbf{a}_i\,(\mathbf{a}_i^T\,\mathbf{a}_j)\end{aligned} \tag{13}$$

Cuando los diversos $\mathbf{a}_i$ son ortogonales entre sí, el segundo término de (13) desaparece y

los vectores $\mathbf{a}_i$ constituyen los autovectores de la matriz **W**. Cuando los $\mathbf{a}_i$ no son ortogonales, (13) puede expresarse como

$$\mathbf{a}'_j = \lambda_j \mathbf{a}_j + \sum_{i \neq j} \varepsilon_{ji} \mathbf{a}_i \tag{14}$$

donde ε_{ji} es la proyección del vector $\mathbf{a}_j$ sobre $\mathbf{a}_i$ ($\mathbf{a}_i^T \mathbf{a}_j$). Aplicando el ***power method*** a $\mathbf{a}'_j$ obtenemos por inducción, para *k* suficientemente grande

$$\mathbf{q}_j = \lambda_j \mathbf{a}_j^{(k)} + \sum_{i \neq j} \varepsilon_{ji} \mathbf{a}_i^{(k)} \tag{15}$$

donde $\mathbf{q}_j$ es el autovector cuya "semilla" es $\mathbf{a}_j$. En otras palabras, cada autovector de **W** es una combinación lineal de los diversos $\mathbf{a}_i$. Puede demostrarse que los autovectores minimizan la "medida de error"

$$E = \sum_{I,\ J} |\mathbf{a}_i^T \mathbf{q}_j| \ / \ (\mathbf{q}_j^T \mathbf{q}_j) \tag{16}$$

En efecto la matriz **W** ha adquirido un nuevo conjunto de autovectores que reemplaza a los vectores de entrenamiento originales. Esta sensibilidad de las matrices de asociación ante la no-ortogonalidad de sus vectores de entrenamiento limita grandemente su uso práctico. Una solución a este problema se plantea en la próxima sección.

3.2. Brain-State-in-a-Box (BSB).

La red neural Brain-State-in-a-Box (Anderson 1977) es un sistema iterativo definido por la siguiente ecuación de recurrencia:

$$\mathbf{a}^{(k+1)} = \sigma\left(\mathbf{W}\, \mathbf{a}^{(k)}\right) \tag{17}$$

donde σ es una función vectorial no lineal saturable de la forma:

$$\sigma(\mathbf{x}) = \begin{cases} \mathbf{x} & \textit{si } |\max x_i| < 1 \\ \mathbf{x} \ / \ |\max x_i| & \textit{si } |\max x_i| \geq 1 \end{cases} \tag{18}$$

El propósito de su peculiar forma se hará evidente más adelante.

La matriz BSB es una matriz autoasociativa, construida de acuerdo al método de la sección anterior, sumada a una matriz identidad. La matriz resultante posee los mismos autovectores que la matriz asociativa original, pero sus autovalores son siempre mayores o iguales a 1. En efecto, si **q** es un autovector de **W**, tenemos

$$(\mathbf{W} + \mathbf{I})\, \mathbf{q} = \mathbf{W}\, \mathbf{q} + \mathbf{I}\, \mathbf{q} = \|\mathbf{q}\|^2\, \mathbf{q} + \mathbf{q} = (\|\mathbf{q}\|^2 + 1)\, \mathbf{q} \tag{19}$$

Los vectores de entrenamiento y los de prueba deben construirse a partir de elementos

tomados del conjunto discreto {-1, 0, +1}, con significado convencional de *falso, desconocido* y *cierto*, respectivamente. No es necesario normalizar los vectores.

La ecuación (17) puede representarse por medio del siguiente diagrama de bloques, similar a los empleados en teoría de control:

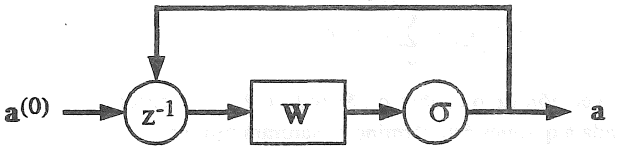


Figura 1

donde el bloque z^{-1} produce un retardo de una unidad de tiempo, con valor inicial $\mathbf{a}^{(0)}$. El vector **a** es el "vector de estado" de este sistema de tiempo discreto.

Los puntos estables no nulos del sistema sólo se hallan en las esquinas del hipercubo definido por la función σ, dondes todos los componentes del vector **a** son +1 ó -1. No pueden haber puntos estables en el interior del hipercubo (excepto en el origen) pues la presencia de autovalores mayores que 1 causa que el circuito posea realimentación positiva, y hace que los componentes del vector de estado crezcan hasta saturarse en alguna esquina. En efecto, las trayectorias del estado del sistema están restringidas al interior de la "caja" definida por la función σ– de allí el nombre de la red. El algoritmo BSB itera hasta que el vector **a** deja de cambiar, por lo que su terminación está garantizada.

En el interior del hipercubo el sistema BSB es lineal, y responde a los métodos del álgebra lineal. Tan pronto uno de los elementos del vector de estado se satura ("tropezándose" con alguna pared de la caja), la función σ se comporta (a excepción del signo) como la función ϕ definida en (7). En efecto, la iteración definida por el algoritmo BSB es muy similar al *power method*, pero con una diferencia importante: BSB produce el autovector más cercano a $\mathbf{a}^{(0)}$, y no necesariamente el autovector dominante de **W**. Supongamos que el vector de prueba es $\mathbf{a}^{(0)}=\sum_i \rho_i \mathbf{q}_i$ (expresado según su proyección sobre la base autovectorial). Aplicando k iteraciones de BSB, y asumiendo que el vector de estado permanece dentro del hipercubo, tenemos

$$\mathbf{a}^{(1)} = \mathbf{W}\,\mathbf{a}^{(0)} = \left(\sum_i \mathbf{q}_i \mathbf{q}_i^T\right)\left(\sum_i \rho_i \mathbf{q}_i\right) = \sum_i \rho_i \lambda_i \mathbf{q}_i \qquad (20)$$

$$\mathbf{a}^{(k)} = \mathbf{W}\,\mathbf{a}^{(k-1)} = \sum_i \rho_i \lambda_i^k \mathbf{q}_i \qquad (21)$$

Podríamos pensar que (21) se aproximará al autovector dominante, como ocurre en el *power method*. Sin embargo, la presencia de σ encierra las trayectorias de **a** en una caja; o dicho de otra forma, el primer autovector cuyo factor $\rho_i \lambda_i^k$ se haga mayor que 1 hará que $\mathbf{a}^{(k)}$ quede dividido por ese mismo factor (debido a (18)). Si el autovector "ganador" es $\mathbf{q}_j$, los factores de los demás autovectores se convertirán en $(\rho_i/\rho_j)(\lambda_i/\lambda_j)^k < 1$. Esta propiedad de generar el autovector más cercano al vector de prueba da a la red BSB cierta inmunidad al ruido. Si descomponemos al vector de prueba en un autovector $\mathbf{q}_j$ y un pequeño término de

error $\mathbf{q}_\varepsilon$ ortogonal a $\mathbf{q}_j$, la iteración BSB nos da

$$\begin{aligned} \mathbf{a}^{(1)} = \mathbf{W}\,\mathbf{a}^{(0)} &= \left(\sum_i \mathbf{q}_i\,\mathbf{q}_i^T\right)(\mathbf{q}_j+\mathbf{q}_\varepsilon) \\ &= \mathbf{q}_j\,\mathbf{q}_j^T\,\mathbf{q}_j + \sum_{i\neq j}\mathbf{q}_i\,\mathbf{q}_i^T\,\mathbf{q}_\varepsilon \\ &= \lambda_j\,\mathbf{q}_j + \sum_{i\neq j}\varepsilon_i\,\lambda_i\,\mathbf{q}_i \end{aligned} \qquad (22)$$

donde ε_i es la proyección de $\mathbf{q}_\varepsilon$ sobre $\mathbf{q}_i$. Si todo $\varepsilon_i\lambda_i < \lambda_j$, entonces sucesivas iteraciones harán que (21) tienda a $\mathbf{q}_j$, pues éste término se saturará primero.

El funcionamiento de la red BSB se puede resumir informalmente de la siguiente manera: dado un vector inicial $\mathbf{a}^{(0)}$, el autovector "más cercano" (con mayor proyección sobre $\mathbf{a}^{(0)}$) guía la evolución de **a** hasta un punto de la "pared" del hipercubo cercano a su esquina. Si $\mathbf{a}^{(0)}$ es idéntico a algún autovector, el estado evolucionará directamente hacia la esquina apropiada. Si hay varios autovectores suficientemente cercanos, cada uno de ellos tratará de guiar independientemente al estado hacia su propia esquina; como resultado, **a** se moverá en una dirección intermedia. El autovector con mayor autovalor triunfará ligeramente, y su ventaja crecerá con cada iteración hasta hacer que **a** se dirija a la esquina asociada con dicho autovector. Si los vectores de entrenamiento con los que se creó la red no son del todo ortogonales, la dinámica BSB forzará de todos modos al estado a evolucionar hacia la esquina apropiada, aunque los autovectores ya no "apuntan" exactamente a las esquinas. Si la condición de ortogonalidad no es violada demasiado, estos "vectores guía" apuntan lo suficientemente cerca de las esquinas como para que la dinámica BSB sea capaz de corregir el error cometido.

4. Extracción de Autovectores con Redes Neurales.

Una simple inspección del diagrama de bloques del *power method* y de la red BSB, así como lo dicho en las secciones anteriores, nos han de convencer que ambos algoritmos son miembros de una misma familia, con propiedades matemáticas similares.

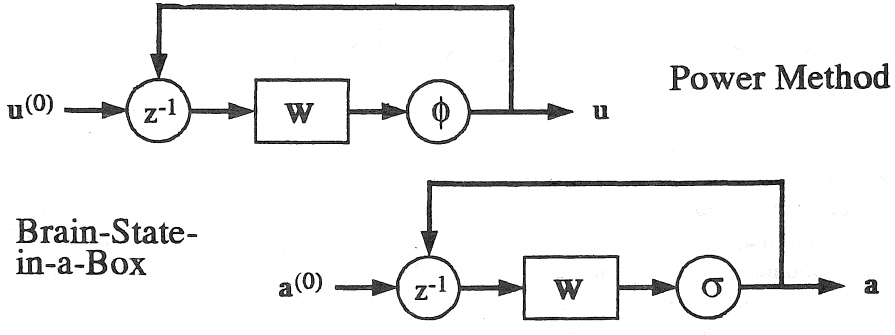


Figura 2

Ambos métodos son iterativos; su convergencia es controlada por ϕ en el caso del *power method* y por σ en el caso de BSB; y la evolución dinámica de su estado es determinada por las propiedades autovectoriales de la matriz **W**. Su diferencia más notable estriba en que BSB es un sistema no-linear, muy sensible a su vector inicial $\mathbf{a}^{(0)}$, mientras que el *power method* es linear y esencialmente insensible a $\mathbf{u}^{(0)}$. Es interesante notar como las debilidades de un método son las virtudes del otro.

Como ya hemos visto, es posible extender el *power method* para obtener los autovectores subdominantes. Nos podemos entonces preguntar, ¿será posible extender BSB para obtener los componentes autovectoriales secundarios de su vector de prueba?

4.1. Realización del método de Aitken.

La respuesta es: sí, mediante la implementación de la metodología de purificación de Aitken. Para comprenderlo, veamos el diagrama de bloques del método de Aitken tal como se aplica al *power method:*

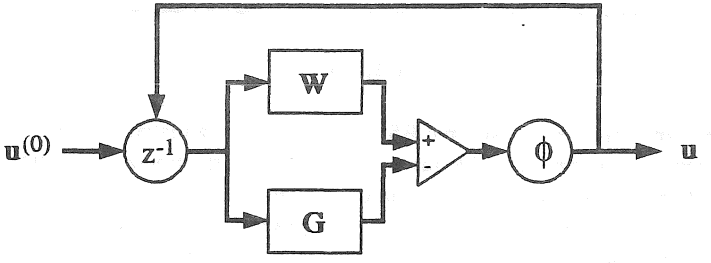


Figura 3

donde la matriz **G** contiene los términos de purificación de (8), que para obtener el k-ésimo autovector son

$$\mathbf{G} = \sum_{i<k} (\lambda_i / \|\mathbf{q}_i\|^2)\, \mathbf{q}_i \mathbf{q}_i^T \qquad (23)$$

Podríamos pensar que lo único necesario para adaptar la Figura 3 al uso de una red BSB es cambiar la función ϕ por σ. Sin embargo no es conveniente implementar directamente el diagrama con redes neurales, pues en efecto nos veríamos forzados a combinar las matrices **W** y **G** en una sóla (como se hace en el método de Aitken original), ya que ambas están íntimamente acopladas a través de la linealidad del bloque sumador. Si combinamos las matrices, el circuito se reduce a una sóla red neural similar a BSB, pero con una regla de aprendizaje difícil de implementar debido a los diferentes requerimientos de los componentes **W** y **G** de la matriz integrada. Los detalles de entrenamiento e interrogación de esta red híbrida son tan engorrosos que la hacen poco práctica.

Puesto que las propiedades de **W** y **G** son diferentes, vale la pena aislarla dichas matrices

en redes neurales independientes. Hay dos restricciones importantes que debemos tomar en cuenta al diseñar el circuito final. La primera es que la matriz **W** deberá formar parte de una red BSB para que pueda cumplir con su parte en el *power method* subyacente. La segunda consiste en que **G** deberá recibir directamente como entrada el vector de estado **u** para que pueda "aprenderlo" por medio de una variación apropiada de (23).

Luego de algunas manipulaciones (en las que se aprovechan las propiedades de z^{-1}, la linealidad parcial de σ, y el hecho que **G** es inicialmente 0) logramos convertir la Figura 3 de acuerdo a las restricciones antes indicadas:

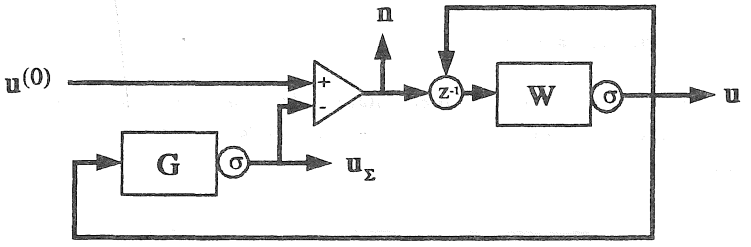


Figura 4

donde se puede apreciar como la sub-red BSB (que incluye a la matriz **W**) fué colocada en el lazo de realimentación negativa del bloque sumador, que en este montaje se comporta como lo haría un amplificador operacional en un circuito electrónico.

Aunque las transformaciones realizadas para pasar de la Figura 3 a la 4 no preservan exactamente la dinámica temporal del sistema original, sí preservan su flujo de información y a la vez introducen nuevas e interesantes propiedades.

4.2. Propiedades requeridas para G.

Un punto crítico del circuito de la Figura 4 es la matriz **G**. Debido a su posición, esta matriz recibe como entrada el autovector principal (extraido por la matriz **W**) contenido en la salida del bloque sumador. Para poder cumplir con su función dentro del método de Aitken, **G** deberá "aprender" este autovector y, por medio de su dinámica no-lineal asociada, reproducirlo constantemente como su salida, para que el bloque sumador pueda cancelarlo de la entrada de la sub-red BSB y así permitirle a ésta extraer el próximo autovector.

Puesto que BSB produjo el autovector canonico, desconocemos con que factor de escala éste se halla en la entrada $\mathbf{u}^{(0)}$ del circuito, por lo que también ha de ser responsabilidad de **G** hallar dicho factor. El método que se eligió para ello es acumular gradualmente el autovector con cada presentación del mismo a la entrada de **G**, de forma tal que su factor de escala a la salida crezca con cada nueva presentación, hasta cancelarlo totalmente en $\mathbf{u}^{(0)}$. Este comportamiento es similar a la carga gradual de un condensador, por lo que podemos decir que la red que incorpora a **G** "integra" los autovectores que recibe como entrada.

A medida que BSB produce secuencialmente los autovectores contenidos en su entrada,

G los debe ir acumulando independientemente, reproduciéndolos todos juntos y con el factor de escala apropiado. Es de hacerse notar que las entradas de G siempre serán ortogonales entre sí por haber sido obtenidas directamente de la salida de la sub-red BSB. Esto es indispensable para que la matriz G pueda mantener la independencia en el almacenamiento y recuperación de los diversos vectores que la componen.

Es necesario que la memoria de G decaiga con el tiempo. De no ser así, el circuito no podrá responder a los autovectores contenidos en los nuevos patrones presentados en $\mathbf{u}^{(0)}$. Sin este decaimiento el circuito quedaría "ciego" a los autovectores del primer patrón presentado, haciéndole imposible responder a ellos en subsiguientes patrones. La constante de tiempo de este decaimiento, similar a la descarga de un condensador, determina la dinámica temporal del resto del sistema y debe ser ajustada para obtener los tiempos de respuesta deseados.

El decaimiento de G, junto con el hecho de hallarse esta matriz en un lazo de realimentación negativa, garantiza además que el proceso de "carga" que genera los factores de escala de los autovectores no prosiga indefinidamente hasta saturar a G: si el factor de escala es demasiado grande, la realimentación negativa tratará de reducirlo hasta el punto exacto en el que el autovector quede anulado en la entrada "positiva" del módulo sumador.

4.3. Implementación de G.

En términos de redes neurales, la matriz G corresponde a una red autoasociativa recurrente, capaz de ser entrenada "en línea." Puesto que sus entradas siempre serán ortogonales entre sí, el entrenamiento puede realizarse sin dificultad usando la regla de Hebb clásica sin peligro de interferencia entre los vectores memorizados.

No cualquier red autoasociativa recurrente puede servirnos en este circuito. Las características dinámicas mencionadas en la sección anterior limitan nuestra elección a dos candidatos: la red *Shunting Grossberg* (Grossberg 1973) y su versión simplificada, la red *Additive Grossberg* (Grossberg 1968). Por simplicidad seleccionamos la red *Additive Grossberg* (que en lo sucesivo denominaremos AG) para implementar las funciones requeridas en G.

Las propiedades de la red AG derivan directamente de las ecuaciones diferenciales vectoriales que la definen. En su forma más general éstas son:

$$\dot{\mathbf{a}} = -\beta\mathbf{a} + \alpha\,\sigma(\mathbf{G}\,\mathbf{a}) + \gamma\mathbf{i} \tag{24}$$

$$\dot{\mathbf{G}} = -\mu\mathbf{G} + \eta\,\mathbf{a}\,\mathbf{a}^{T} \tag{25}$$

donde σ es una función sigmoide de la forma

$$\sigma(x) = (1 + e^{-\omega x})^{-1} \tag{26}$$

La ecuación (24) determina el comportamiento dinámico de la red ante los estímulos. En particular, el factor β determina la rata de decaimiento exponencial del vector de actividad neural **a**; el factor α controla el grado de reverberación causado por la matriz autoasociativa G, lo cual define cuan sostenida es la respuesta intrínseca de la red a los múltiples vectores

almacenados en G; y el factor γ determina la constante de tiempo de "carga" de la actividad ante la estimulación causada por el vector de entrada **i**.

La ecuación (25), por su parte, define la regla de aprendizaje de la red AG, que es un caso particular de la ecuación (23) llevada al límite contínuo. Los factores μ y η determinan el grado de "plasticidad" de la memoria de la siguiente manera: μ regula la constante de tiempo de decaimiento de la memoria, y η controla la rata de aprendizaje Hebbiano; entre ambos definen el comportamiento integrativo deseado en **G**. La necesidad de balancear estos dos factores provoca lo que Grossberg denomina el "dilema plasticidad vs. estabilidad"; es decir, la facilidad con la cual **G** aprende nuevos vectores vs. la durabilidad de los vectores aprendidos. No es posible tener plasticidad y estabilidad a la vez: es necesario establecer un compromiso. En nuestro caso el compromiso se resuelve en favor de la plasticidad, convirtiendo en ventaja lo que para la red AG aislada es un defecto.

Este no es lugar para realizar un estudio detallado de las propiedades dinámicas del sistema de ecuaciones (24)-(25). Por una parte, Amari, Cohen y Grossberg han realizado análisis profundos del comportamiento de una familia de redes que incluye a la red AG como caso particular (Amari 1972, Cohen y Grossberg 1983).

Por otra parte, no es necesario estar familiarizado con todas las peculiaridades de la red AG, ya que en realidad sólo nos interesan aquellas características que ya hemos descrito en secciones anteriores. El hecho de que la red AG esté situada en un lazo de realimentación negativa minimiza el impacto que puedan tener sobre el circuito sus desviaciones con respecto al comportamiento "ideal" deseado. Este fenómeno de "idealización de características" provocado por la realimentación negativa es bién conocido y aprovechado en el diseño de circuitos electrónicos.

5. Comportamiento Dinámico del Sistema.

El comportan\miento dinámico de este circuito es muy interesante, pero a la vez muy difícil de describir formalmente. Aquí tan sólo intentaremos dar una idea de lo que ocurre a medida que los patrones son analizados. Un futuro artículo examinará más detalladamente estos aspectos mediante técnicas gráficas descriptivas (similares a un "osciloscopio simbólico") que aún están en desarrollo.

5.1. La acumulación de autovectores.

Como ya fué dicho, la red AG acumula (o integra) las diversas respuestas de la red BSB. Por una parte lo hace a través de la modificación de la matriz **G** (lo que Grossberg denomina *long-term memory*), y por otra parte a través de los términos que contienen α y β en la ecuación (24). Este *short-term memory* (así denominado por Grossberg) "sujeta" el autovector el tiempo suficiente como para memorizarlo en el *long-term memory*, y a la vez sirve para combinar linealmente los autovectores a los cuales la red BSB ya ha respondido.

Según hemos mencionado, tanto el *short-term memory* como el *long-term memory* deben decaer para que la red AG pueda responder a nuevos autovectores. El efecto secundario

de este decaimiento es el de permitir a la red AG hallar el factor apropiado para cancelar el autovector: cada presentación de éste "carga" la red, mientras que el factor de decaimiento la "descarga" contínuamente. Gracias a la realimentación negativa, la combinación de ambos procesos estabiliza a la red en el factor correcto para cada autovector memorizado.

5.2. La generación serial de autovectores.

La red BSB responde inicialmente al vector total de entrada del sistema produciendo el autovector más cercano (o el de autovalor más grande, si hay varios igualmente cercanos). Dicho autovector es acumulado en la red AG y cancelado de la entrada de BSB por medio del módulo sumador; entonces BSB responde al siguiente autovector contenido en su entrada, y así sucesivamente. En efecto, la red BSB produce en secuencia (en $\mathbf{u}$, Figura 4) los autovectores que componen al estímulo presentado en $\mathbf{u}^{(0)}$ en orden decreciente de su importancia y autovalores.

El circuito, entonces, responde a un estímulo estático causando la rápida generación de la secuencia de autovectores que lo componen (por la red BSB) y la acumulación de dicha secuencia (por la red AG). Todo ello gracias a la dinámica de esta última, controlada principalmente por los factores α y η. Pero, debido a esa misma dinámica, la recién producida secuencia se repite lentamente una y otra vez a medida que la red AG "olvida" algún autovector bajo el control de los factores β y μ. Esas secuencias "refrescan" los factores de escala de los autovectores que han decaido, manteniéndolos estables dentro de cierto margen de tolerancia determinado primariamente por γ. A este contínuo proceso de "refrescado" de la memoria lo denominaremos *régimen estático de functionamiento.*

Al cambiar el patrón de entrada del circuito, la red AG no responde inmediatamente, sino que su salida cambia lentamente (bajo el control de β y α). Si los cambios del patrón son más rápidos que esa constante de tiempo, éstos no serán reflejados en la salida de la red AG, aunque su autovector principal sí será extraido por la red BSB. A esta forma de respuesta ante estímulos transitorios la denominaremos *régimen dinámico de funcionamiento.* Si el patrón se mantiene por tiempo suficiente, la red AG alcanzará a cambiar su salida al autovector principal y el sistema abandonará el régimen transitorio y entrará en el régimen estático.

5.4. El vector de novedad.

Este vector (denominado $\mathbf{n}$ en la Figura 4) es la diferencia entre el vector $\mathbf{u}^{(0)}$ de entrada del circuito y el vector $\mathbf{u}_{\sum}$ de salida de la red AG, y contiene todo aquello que aún no ha sido reconocido por la red BSB. En particular, debido al retardo de la respuesta (o "inercia") de la red AG, los cambios repentinos del input se reflejan inmediatamente en el vector de novedad; tan sólo cuando la red AG ha incorporado la respuesta de la red BSB, ésta desaparece del vector de novedad. Debido al proceso de carga y descarga de la red AG, esta desaparición no es instantánea, sino que sigue una curva exponencial.

El vector de novedad puede ser, por lo tanto, una fuente importante de datos para un posible "sistema de control de atención" en cuanto le puede proveer de todo aquello que no ha podido ser identificado, así como de aquellos estimulos que cambian rápidamente.

6. Conclusiones.

El análisis de patrones superpuestos o ambiguos es un tema difícil para cualquier técnica de computación. Sin embargo, esta situación es muy común en la vida real, y los cerebros tanto del hombre como de los animales la manejan generalmente muy bién. Dotar a las redes neurales de una herramienta que las pueda ayudar en esa tarea nos parece bastante importante.

La capacidad del presente circuito de discriminar los componentes que conforman una muestra desconocida y presentarlos uno a uno, en orden decreciente de importancia, es un paso hacia la integración de la técnica de las redes neurales con los métodos tradicionales de razonamiento simbólico. En efecto, analizar algo desconocido en términos de lo ya conocido, y hacerlo contemplando todas las semejanzas a lo largo de todos los ejes en un análisis multidimensional, es lo propio de la asimilación intelectual. Hacer menos es meramente responder ciegamente a estereotipos.

Los elementos matemáticos utilizados nos sirvieron primero de inspiración y luego de herramienta para el análisis del comportamiento del montaje. Pensamos que, en general, los métodos iterativos del álgebra lineal podrían beneficiarse mucho de las técnicas de "ensayo y error" y corrección de errores propias de las redes neurales, así como de la técnica de anulación de errores en lazos cerrados con realimentación negativa que empleamos en este circuito.

El comportamiento dinámico del circuito es difícil de describir, y amerita un estudio más profundo y exaustivo: aún no se han desarrollado buenos "osciloscopios simbólicos" capaces de mostrar en forma visualmente significativa trayectorias complejas en un espacio multidimensional, por lo que lamentablemente las técnicas simplificadas de análisis, tan usadas en electrónica para reducir a niveles manejables la complejidad de un circuito, no son directamente aplicables a este caso. Nuestros estudios preliminares, sin embargo, nos permiten poner grandes expectativas en las capacidades del circuito presentado, así como en circuitos similares.

Referencias.

Aitken, A. C. (1937). **The evaluation of the latent roots and vectors of a matrix**, *Proceedings of the Royal Society of Edinburgh, Section A*, **57**, 269-304.

Amari, S-I. (1972). **Characteristics of random nets of analog neuron-like elements**, *IEEE Transactions on Systems, Man, and Cybernetics*, **2**, 643-657.

Amari, S-I. (1977). **Neural theory of association and concept formation**, *Biological Cybernetics*, **26**, 175-185.

Anderson, J., Silverstein, J., Ritz, S., Jones, R. (1977). **Distinctive features, categorical perception, and probability learning: Some applications of a neural model**, *Psychological Review*, **84**, 413-415.

Cohen, M., Grossberg, S. (1983). **Absolute stability of global pattern formation and parallel memory storage by competitive neural networks**, *IEEE Transactions on Systems, Man, and Cybernetics*, 13, 815-826.

Fukushima, K. (1980). **Neocognitron: A self-organizing neural network for a mechanism of**

pattern recognition unaffected by a shift in position, *Biological Cybernetics*, 36, 193-202.

Fukushima, K., Miyake, S., Takayuki, T. (1983). **Neocognitron: A neural network model for a mechanism of visual pattern recognition**, *IEEE Transactions on Systems, Man, and Cybernetics*, 13, 826-834.

Grossberg, S. (1968). **Some nonlinear networks capable of learning a spatial pattern of arbitrary complexity**, *Proceedings of the National Academy of Sciences*, 59, 368-372.

Grossberg, S. (1973). **Contour enhancement, short-term memory, and constancies in reverberating networks**, *Studies in Applied Mathematics*, 52, 217-257.

Jennings, A. (1977). **Matrix Computation for Engineers and Scientists**, *John Wiley & Sons, Chichester*.

Multiple Fonts from a Single Font Optical Character Recognizer

Riccardo Ferrari, Malvina Duarte

Laboratorio de Electrónica Digital
Universidad Católica "Nuestra Señora de la Asunción"
Po. Box 1718
Asunción - Paraguay
email: rferrari@ledip.uucp, mduarte@ledip.uucp

Carlo Viscomi

Hypertex User Group
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
Milano - Italia

Some available OCR systems provide documents with a single font, style and syze. The present work studies the way for producing documents with multiple fonts, style and size starting from a trainable, but single font, recognizer. The analysis we developped led to the definition a *recognizing methodology*. This methodology has been ipmlemented in a prototype that we describe in the paper. Finally we present an example of application to a Calculus textbook.

Algunos sistemas disponibles para la lectura óptica de documentos de papel dotados de reconocimiento de caracteres suministran documentos con un único tipo de carácter, estilo y cuerpo. En el presente trabajo se estudia como producir documentos con tipo de carácter, estilo y cuerpo múltiples, utilizando un reconocedor instruible de este tipo. El análisis del problema que hemos efectuado nos ha llevado a definir una *metodología de reconocimiento* que hemos implementado en el prototipo que presentamos. Al final de la presente nota se describe un ejemplo de aplicación de la metodología propuesta sobre un texto de análisis matemático.

Introducción

Para simplicidad de referencia, comenzamos introduciendo algunos términos. Con el término *símbolo* se entiende un elemento cualquiera del código ASCII y con *atributos* el tipo de carácter, de estilo y de cuerpo de un carácter.
La metodología de reconocimiento está estructurada en dos principales actividades. En la primera se efectúa un análisis preliminar de los símbolos a reconocer, de aqui se determina una clasificación en relación a los atributos de los cuales están dotados, se instruyen al reconocedor y al prototipo en base a la clasificación definida. La clasificación se basa en la agrupación de símbolos que presentan un atributo común. Luego se efectúa el reconocimiento con producción de documentos de tipo texto, cuyos símbolos presentan atributos comunes, es decir un único tipo de carácter, estilo y cuerpo. La segunda actividad consiste en la importación en el prototipo de estos documentos y en la conversión automática de los atributos de los símbolos sobre la base de la clasificación. El resultado que se obtiene es la reproducción en formato electrónico del texto sobre papel, fiel en todos los símbolos respecto a sus atributos originales.
El prototipo se presenta en su aspecto funcional y en las modalidades de uso de cada comando.
El artículo comienza con una breve descripción de los reconocedores de caracteres, clasificados en relación a funcionalidades y modalidades de uso.

1. Digitalización de un texto

1.1. Los reconocedores de caracteres

Los reconocedores (Optical Character Recognizer - OCR) se pueden dividir en tres grandes categorías: No entrenables, Entrenables, Automáticos.
Los OCR de tipo *No entrenable* se basan en las confrontaciones de conjuntos de modelos, llamadas clases de referencia. Tales conjuntos reproducen los símbolos a reconocer y son proveidos sólo para algunos atributos de los símbolos. En consecuencia, resultan reconocibles solo las impresiones que utilizan los mismos simbolos y solo con los atributos conocidos.
En los OCR de tipo *Entrenable*, las clases de referencia, llamadas "type table", son construidas por el usuario en un diálogo con el reconocedor el cual propone, una a la vez, áreas de bit-map que considera ser simples símbolos, en relación a las

cuales es posible introducir vía teclado el símbolo en código ASCII que la bit-map representa en forma gráfica. Tal posibilidad ha permitido gestionar los diferente atributos, como será detalladamente especificado a continuación.
Los OCR de tipo *Automático* están en grado de reconocer automáticamente, esto es sin ninguna fase preliminar de instrucción, textos caracterizados por una gran variedad de atributos de los símbolos, también en el interior del mismo documento.

1.2. Metodología de reconocimiento

El universo U del discurso es compuesto por todos los posibles símbolos s utilizables en cualquier documento de tipo textual. Para cada elemento de U se definen cuatro propriedades a(s), t(s), c(s), e(s), que describen respectivamente el valor de codigo ASCII[1] y los atributos: tipo (font), cuerpo (size) y estilo (style) del símbolo mismo. Se consideran símbolos diferentes aquellos que difieran por una o más de las propiedades.
Los símbolos que tienen mismo código ASCII constituyen las clases de una relación de equivalencia sobre U. Indicamos con ASCII* el conjunto de todas las clases de ASCII-equivalencia.
Considerado un documento de tipo texto (indicado con t), que consiste en una particular secuencia de símbolos, denotamos con S el conjunto de los símbolos que aparecen en t. Se definen atributos del documento las propiedades tipo, estilo y cuerpo (indicados respectivamente con t(t), c(t) y c(t)) de la mayoria de sus símbolos. En la operación de asignación inicial se indican al reconocedor los valores de los atributos del documento. Estos atributos serán asignados a todos los símbolos del documento reconocido.
Con el término símbolo general se entiende un símbolo cuyos atributos coinciden con aquellos del documento al que pertenece. Se define conjunto de los Símbolos Generales SG:

$$S_G = \{ s \mid s \in S, (t(s) = t(t)) \cap (c(s) = c(t)) \cap (e(s) = e(t)) \}.$$

Con símbolo especial se identifica un símbolo que posee por lo menos un atributo diferente de los atributos del documento al cual pertenece. El conjunto de los Símbolos Especiales SE está definido como:

$$S_E = \{ s \mid s \in S, (t(s) \neq t(t)) \cup (c(s) \neq c(t)) \cup (e(s) \neq e(t)) \}.$$

---

1 Consideramos aqui el código ASCII en versión extendida, osea con 256 caracteres

En la Fig. 1.1 se muestra un ejemplo de símbolo especial:

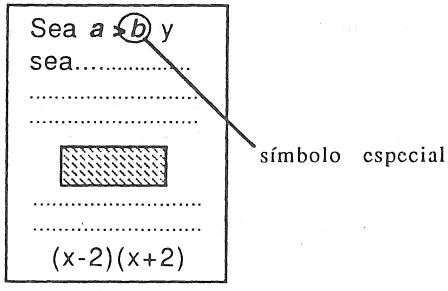


Fig 1.1: Ejemplo de símbolo especial

Con símbolo de pasaje se entiende una secuencia de dos símbolos, de los cuales el primero es el representante de una clase de equivalencia ASCII que no aparece en el texto t y cuyos atributos sean los del texto, y el segundo es un símbolo general. El conjunto de los Símbolos de Pasaje S_P resulta así definido:

$$S_P = \{pg \mid p \in (U\text{-}S_G),\ g \in S_G,\ [(t(p) = t(t)) \cap (c(p) = c(t)) \cap (c(p) = c(t))]\}.$$

Podemos elegir p entre todos los elementos de $(U\text{-}S_G)$ que además compartan los atributos del texto, entonces tenemos $(|ASCII^*| - |S_G|)$ diferentes candidatos a ser posibles p.
Resulta que si consideramos la cardinalidad del conjunto S_P, ésta es dada entonces por:

$$|S_P| = (|ASCII^*| - |S_G|) \cdot |S_G|$$

y representa la cota máxima de símbolos de pasaje y, como veremos a continuación, también de símbolos especiales manejables por el sistema.
La metodología de reconocimiento definida está constituída por la siguiente secuencia de fases:

1) búsqueda en el texto sobre papel de todos los símbolos especiales y creación del "Conjunto Especial" cuyos elementos a la vez son conjuntos. En efecto el "Conjunto Especial" C_E está definido como:

$$C_E = \{ C_{E1}, C_{E2}, \ldots, C_{En} \}$$

donde n es el número de conjuntos especiales encontrados en un documento, y donde el Conjunto Especial k-ésimo C_{Ek} es el conjunto cuyos elementos son símbolos especiales caracterizados por diferentes valores ASCII e iguales atributos de tipo, estilo y cuerpo.
En la figura 1.2 está esquematizado el paso 1.

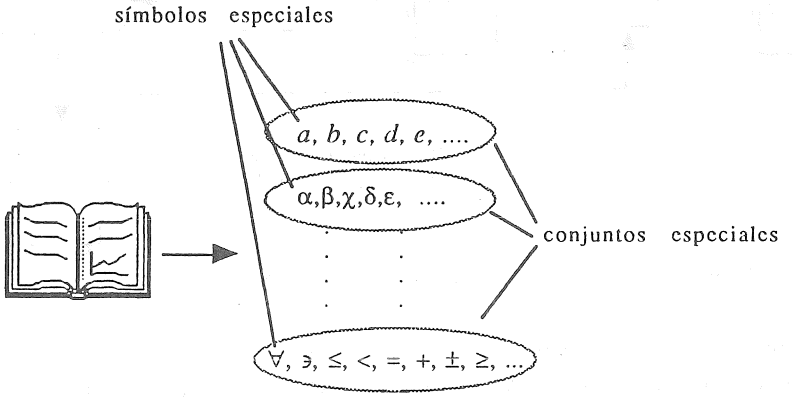


Fig. 1.2: Identificación de los símbolos especiales y definición de los Conjuntos Especiales

2) para cada símbolo especial de cada conjunto especial se procede a la creación de las asociaciones "*símbolo especial → símbolo de pasaje*" como es mostrado en la figura 1.3. Estas asociaciones están definidas por la función biunívoca *Asociación* A:

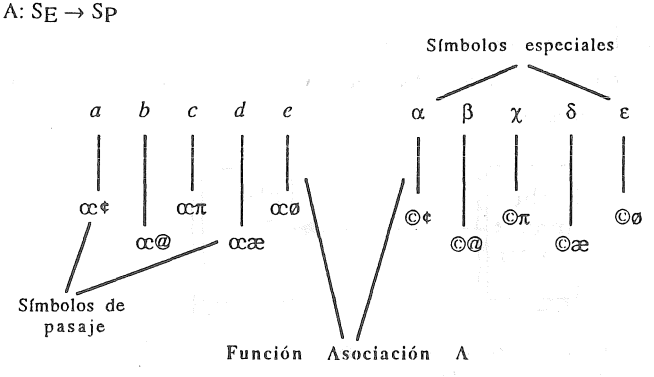


Fig. 1.3: Definición de las asociaciones "símbolo especial Æ símbolo de pasaje" y de la función "Asociación" Λ

3) instrucción de un OCR de tipo entrenable con creación de una *type table* sobre los atributos de los símbolos generales y especiales, utilizando la función Asociación antes definida; reconocimiento del texto (fig. 1.4).

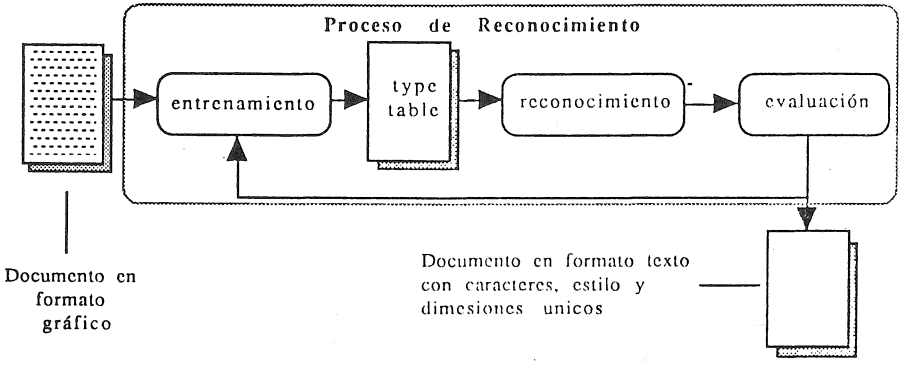


Fig. 1.4: Reconocimiento de texto con OCR de tipo entrenable

4) instrucción del instrumento sobre las asociaciones anteriormente definidas en sentido inverso (*símbolo de pasaje* → *símbolo especial*), definiendo la Asociación Inversa:

$$A^{-1} = (A)^{-1}$$

que es la función inversa de la función Asociación, obviamente también biunívoca:

$$A^{-1}: S_P \rightarrow S_E$$

5) sustitución global de los símbolos de pasaje en los respectivos símbolos especiales utilizando la función Asociación Inversa. El resultado final está representado en la figura 1.5:

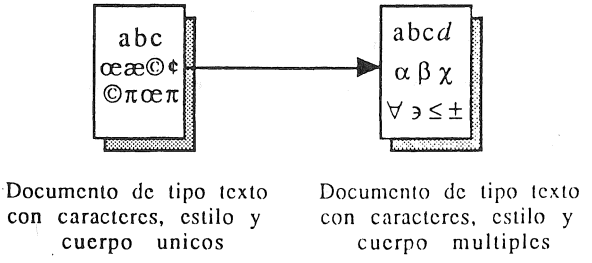


Fig. 1.5: Conversión símbolo de pasaje → símbolo especial

Las fases hasta ahora descritas pueden ser esquemáticamente resumidas a continuación, con la introducción de algunas ulteriores definiciones e introduciendo el operador *clausura* (*), que indica la concatenación de una secuencia, nula, finita o infinita, de símbolos pertenecientes al mismo conjunto al cual pertenece el símbolo que tiene por argumento.

texto inicial t_i, texto en formato de papel impreso constituido por una secuencia de símbolos generales y de símbolos especiales:

$$t_i = \{ (s)^* \mid s \in S_G \cup S_E \}; S_E$$

texto de pasaje t_p, texto en formato electrónico constituído por la secuencia de símbolos generales y de símbolos de pasaje:

$$t_p = \{ (s)^* \mid s \in S_G \cup S_E \};$$

texto final t_f, texto en formato electrónico, constituído por la secuencia de símbolos generales y de símbolos especiales:

$$t_f = \{ (s)^* \mid s \in S_G \cup S_E \};$$

Transformación Inicial T_I, proceso de reconocimiento, guiado por la metodología de reconocimiento , que a partir de un texto inicial y de los conjuntos asociaciones ya definidas, produce el correspondiente texto de pasaje:

$$T_I : t_i \times A \rightarrow t_p;$$

Transformación Final T_F, proceso que a partir de un texto de pasaje y de los conjuntos de las asociaciones inversas encima definidas, obtiene el correspondiente texto final:

$$T_F : t_p \times A^{-1} \rightarrow t_f.$$

El esquema de flujo de la metodología general está dado en la fig. 1.6:

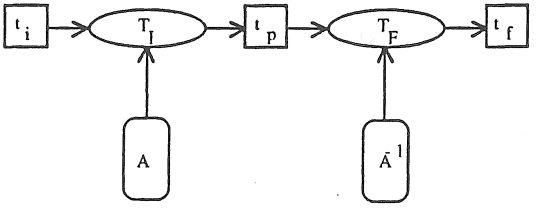


Fig. 1.6: metodología de reconocimiento

2. La estructura funcional del instrumento

La estructura funcional de base del instrumento es dada por la siguiente figura:

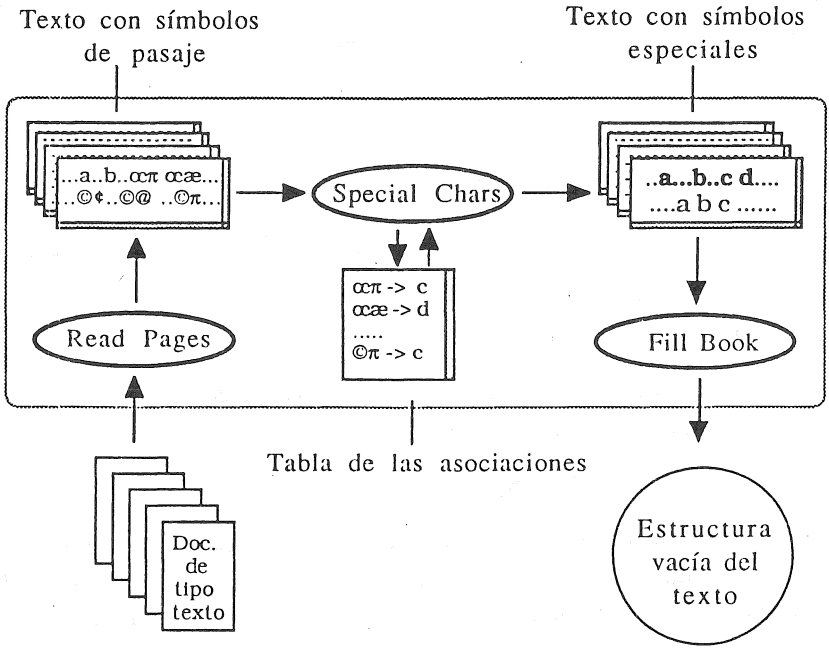


Fig. 2.1: Estructura funcional básica del sistema

El diseño muestra las funcionalidades principales y el tipo de documento en ingreso y en salida del sistema.

Las funcionalidades pueden ser reagrupadas según dos principales actividades: importación de documentos de tipo texto en el sistema (función "Read Pages"); definición e inserción de la función Asociación Inversa con construcción de las Tablas de asociaciones y sustituciones de los símbolos de pasaje en los símbolos especiales correspondientes (función "Special Chars"); exportación de los documentos en una estructura vacía para la realización de un texto electrónico (función "Fill Book").

La fig. 2.1 muestra la secuencia de operaciones de una sesión completa de trabajo; no está definido un orden de ejecución temporal entre las distintas fases,

mientras existe una serie de controles menores sobre el uso correcto del sistema por parte del usuario.

3. Modalidad de uso del tool.

3.1. Operaciones generales

Es necesario especificar al inicio de cada sesión de trabajo el *pathname* de la carpeta de la cual leer los documentos de tipo texto, por el contrario asume el valor de default que corresponde a la última especificación dada.
Es posible escoger el atributo carácter para los documentos presentes en el sistema, que volverá a ser en consecuencia el carácter para el texto general. Tal funcionalidad se puede llamar en cada instante durante la sesión de trabajo, teniendo presente, sin embargo, que la elección, o el cambio de carácter conlleva la pérdida de eventuales símbolos especiales ya restablecidos, puesto que la nueva elección es aplicada sobre el texto de cada documento presente en el sistema.

3.2. Lectura de los documentos (Read Pages)

Un documento de tipo texto, a fin que pueda ser importado en el sistema, debe tener por nombre un número. La elección ha sido dictada por la comodidad del uso, en el caso se tratan de reconocer un texto con numerosas páginas, ya que el instrumento no requiere el *pathname* de los documentos particulares sino más bien informaciones de carácter más general. En efecto la carga de los documentos es hecha por grupos de páginas, indicando la primera y última página del grupo mismo. El grupo, en el interior del sistema, puede ser indentificado con un capítulo nuevo (*Capitolo Nuovo*), en ese caso es necesario indicar también el nombre del capítulo, o bien puede ser insertado en el interior de un capítulo existente (*Nuova Pagina...*) después de la página corriente. Es posible remover una página (*Cancella Pagina...*), o bien un capítulo (*Cancella*), por vez.
La navegación entre los documentos leídos es de tipo secuencial, función permitida a travéz de una *window* de tipo "palette". Es posible navegar a nivel de capítulos, o de páginas en el interior del mismo capítulo, por un elemento (página o capítulo) al precedente o sucesivo, al primero o al último. Están presentes indicaciones sobre la página y capítulo corriente para dar una mínima orientación al usuario.

Finalmente está presente una función que permite de posicionarse directamente sobre una página (*Go to...*).

Las funcionalidades descritas están representadas en la figura 3.2.1.

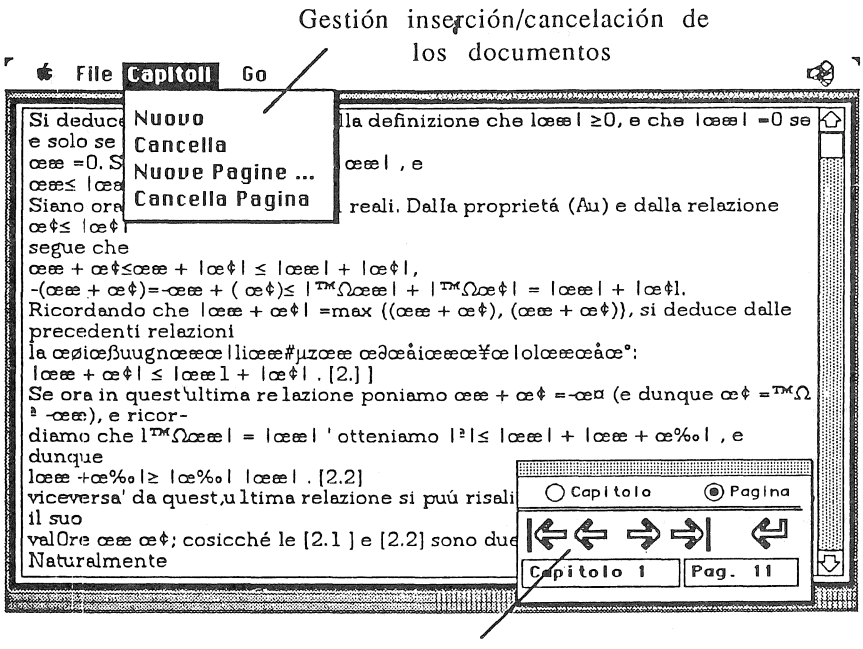


Fig. 3.2.1: La lectura y la navegación de los documentos

3.3. El tratamiento de los caracteres especiales (Special Chars)

Las funcionalidades disponibles en el tool permiten de insertar las asociaciones símbolos de pasaje->símbolos especiales y de aplicar las conversiones en base a los mismos en los documentos presentes en el sistema.

Han sido considerados 46 símbolos especiales tomados del código ASCII extendido, con carácter "Bookman", estilo normal y cuerpo 12, y a cada uno de los cuales viene reservada una "Tablas de Asociaciones" para las asociaciones pertenecientes al grupo correspondiente.

En tales hipótesis el número de asociaciones máximas obtenibles está dado por 210 (o sea 256, la cardinalidad del código ASCII extendido, sustraído de 46, reservados como símbolos de pasaje), multiplicado por 46, el número máximo de símbolos iniciales establecidos para los símbolos de pasaje. El resultado resulta ser 9660 posibles asociaciones, o sea 9660 símbolos especiales gestionables por el sistema. La experiencia muestra que éste es un número más que suficiente para tratar las partes textuales de textos también científicos, normalmente caracterizados por una vasta heterogeneidad de atributos. La figura 3.3.1 muestra la "Symbols Table", o sea la tabla índice de los símbolos especiales, que permite la selección de uno de ellos a fin de posicionarse en la tabla de inserción que se le ha asociado.

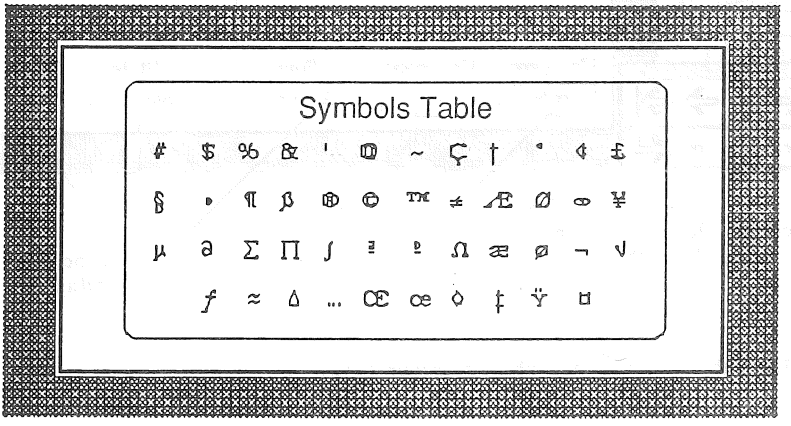


Fig. 3.3.1 : Tabla generál de los 46 símbolos especiales con atributo de caracter "Bookman"

La figura 3.3.2 presenta la tabla correspondiente al símbolo de pasaje inicial "œ".

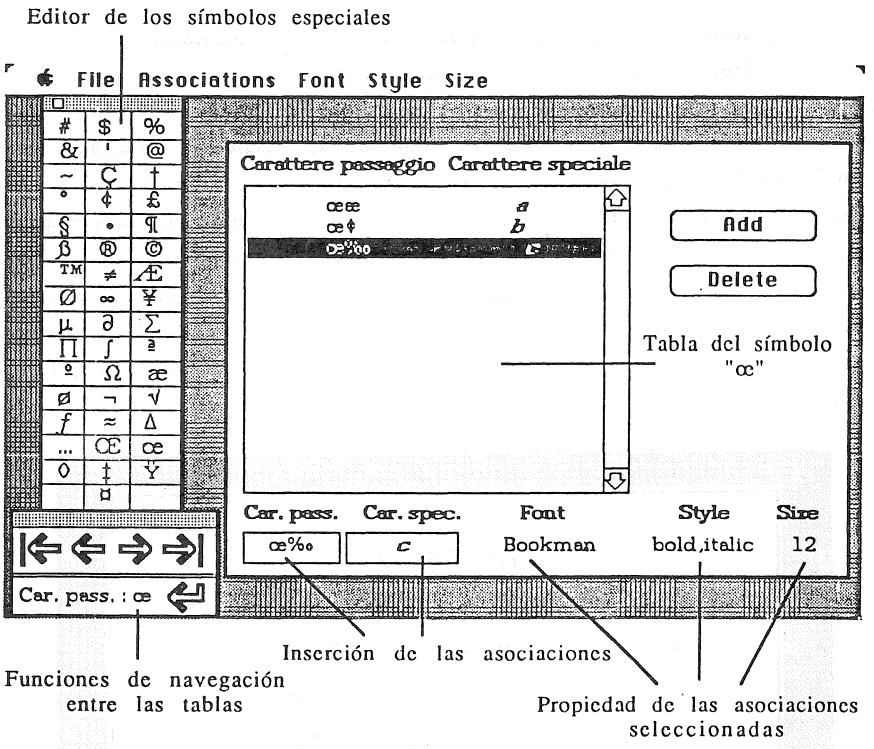


Fig. 3.3.2: Tabla asociadas al símbolo de pasaje inicial "œ".

La inserción de las asociaciones (*Add*) es a través del teclado con el auxilio de un Editor para los símbolos especiales (*AssociationEditor*), a fin de simplificar y volver más rápidas las operaciones; las cancelaciones de las asociaciones está permitida para aquella actualmente seleccionada (*Delete*) o bien para todas las presentes en el sistema (*Delete All*).

Es posible moverse entre las tablas secuencialmente, de la presente a la sucesiva o precedente, o bien a la primera y última del sistema, y finalmente jerárquicamente llamando nuevamente la Symbols Table (*Association Table*), y seleccionando el símbolo de pasaje inicial al cual está asociada la tabla buscada.

Para la fase de sustitución de los símbolos de pasaje en los símbolos especiales, está disponible una serie de funcionalidades que permite de operar sobre la asociación seleccionada (*Apply Selection*), sobre todas las asociaciones de la tabla corriente (*Apply Group*), sobre todas las asociaciones de todas las tablas (*Apply*

All). En cada caso, las sustituciones se realizan sobre todos los documentos presentes en el sistema. La figura 3.3.3 nos muestra las alternativas funcionales descritas:

Fig. 3.3.3: Funciones para la gestión de las asociaciones

Al término de las sustituciones, se suministran algunos datos sobre el número de sustituciones efectuadas por cada asociación, así como sobre el sistema general. La fig. 3.3.4 muestra un ejemplo de sustitución sobre un grupo de 10 páginas importadas en el sistema.

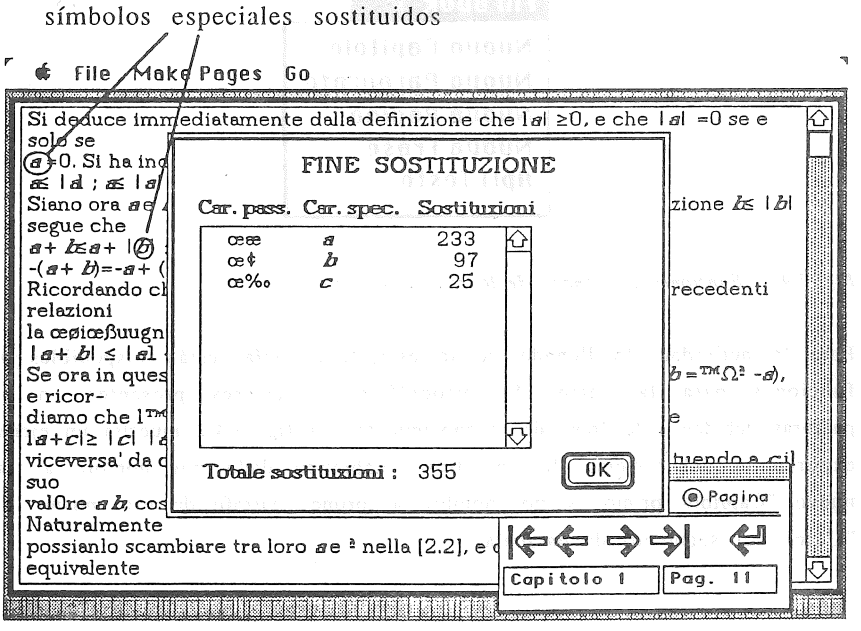


Fig. 3.3.4: Informaciones estadísticas sobre operaciones de sustituciones

3.4. Producción de un texto electrónico (Fill Book)

Un ejemplo del uso final del sistema está dado por la creación del texto electrónico sobre la base de los documentos reconocidos.
En el instrumento se cuenta con funciones que permiten el transporte automático de los documentos presentes en el sistema, en la predispuesta estructura vacía de un texto electrónico.
El texto seleccionado en el documento corriente por el autor puede ser transportado como frase (*Nuova Frase*). La operación de exportación de una frase puede ser precedida por instrucciones para la creación en el texto de un nuevo capítulo (*Nuovo Capitolo*), de un nuevo parágrafo (*Nuovo Paragrafo*) o de una nueva página (*Nuova Pagina*): en los dos primeros casos es posible especificar el nombre del capítulo y del parágrafo que se utilizarán para la orientación del lector del texto. Diversamente, el copiado de una frase se desarrolla según la modalidad de la última efectuada. Además es posible visualizar el estado actual del texto electrónico pidiendo de ingresar en simple lectura (*Apri Testo*).
La fig. 3.4.1 nos muestra las alternativas de la funcionalidad descrita.

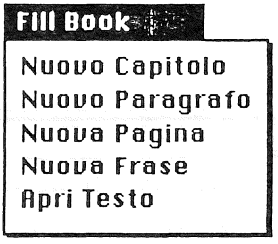


Fig. 3.4.1: Funciones de llenado de la estructura vacía

Para la actividad de llenado de la estructura vacía, están disponibles algunas funciones para la corrección ortográfica de errores presentes en algunas palabras debidas a la frase de reconocimiento. La fig. 3.4.2 muestra un ejemplo de exportación del sistema de una frase, habiendo dado por nombre al capítulo nuevo "Capitolo primo" y por nombre al primer párrafo de este mismo capítulo "L'area del segmento di parabola".

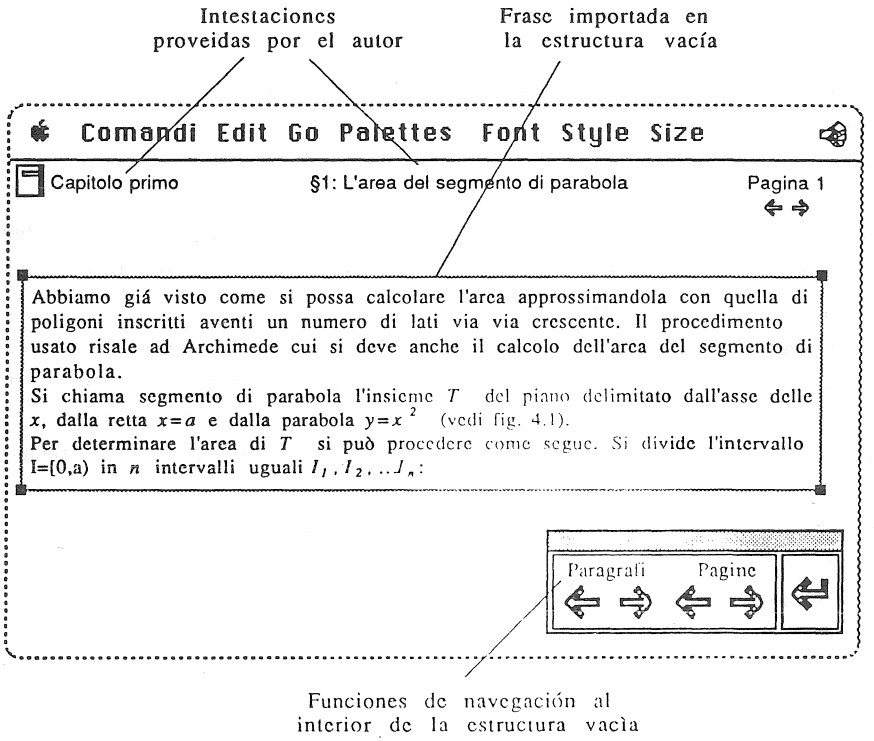


Fig. 3.4.2: Ejemplo de importaciòn de una frase

4. Una experiencia de digitalización

Ha sido conducida una experiencia de digitalización sobre el texto "Analisi Matematica I", autor Enrico Giusti, adoptado en el curso de Doctorado de Ciencia de la Información de la Universidad de Milán. El texto se ha prestado en modo particular al análisis del problema del "data-entry" puesto que presenta una vasta heterogeneidad de atributos, caracteres, estilos, cuerpos, además en número elevado.

Durante el proceso de reconocimiento se ha construído una *type table* que presenta un conjunto de 256 bip-map diversas, de los cuales 107 símbolos de pasaje; por cada símbolo a reconocer han sido hechas más de una entrada, o bien más relaciones bit-map de un mismo símbolo; de 15 normalmente a 40, a fin de aumentar el nivel de reconocimiento. Se han contado así en total 1122 entradas.

En relación a los tiempos de instrucción del reconocedor, ha sido necesario cerca de cuatro días de trabajo, mientras que para el reconocimiento se han estimado

valores de 2 minutos por página normal (en el orden de 1500 caracteres) hasta 4 minutos por página de alta densidad de texto, aquellas del Giusti que contienen las noticias históricas (3400 caracteres). El reconocimiento se realiza en modalidad *batch*: se han soportado sesiones de trabajo con sesenta documentos en entrada sin problemas.

Es de notable importancia construir *type tables* completas y precisas, que trabajen con un nivel alto de reconocimiento, puesto que así se reduce drásticamente la intervención manual de corrección de los errores que comunmente un OCR genera.

Una vez reconocidos todos los 289 documentos, se ha utilizado el prototipo presentado en el párrafo anterior: se han leído los documentos en tiempos del orden de la decena de minutos; se han insertado los grupos de asociaciones en las respectivas tablas de inserción y se ha operado la conversión de las asociaciones. Esta última actividad ha requerido tiempos de ejecución en el orden de 15 horas para un total de cerca de 45.000 sustituciones, calculando tiempos medios de 1,2 segundos por sustitución. En la figura 4.1 está representado un ejemplo de expresión matemática reconocida.

$$\varphi = 2\varphi_{[-1,0)} + 2\varphi_{[0,1)} - \varphi_{[1,2)}$$

Fig. 4.1: Ejemplo de expresión matemática reconocida

Agradecimientos

El presente trabajo ha sido realizado en el ámbito de la cooperación entre el Laboratorio de Electronica Digital (LED) de la Universidad Católica "Nuestra Señora de la Asunción" y el Hypertex User Group (HUG) del Departamento de Ciencia de la Información de la Universidad de los Estudios de Milan, dentro del proyecto "Personal University".

Los autores agradecen al Ministerio de la Universidad e Investigación Científica de Italia y a la Universidad Católica "Nuestra Señora de la Asunción" del Paraguay por los medios puestos a disposición.

Se agradece a Gianni Degli Antoni y Francesca Greselin por las valiosas sugerencias y a todos los miembros del LED y del HUG por la colaboración.

Personal University: Building an Academic Environment

Giovanni Degli Antoni

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
Italia

Riccardo Ferrari

Facultad de Ciencias y Tecnología
Universidad Católica "Nuestra Señora de la Asunción"
Paraguay

The introduction of ever more powerful Information Technologies (IT) will make possible radical changes in the higher instruction field. Portable and low cost workstations can support all academic activities, substituing in a single unit most of the traditional communication technologies (black-board, papers, pencils, books...). The translation of a complete academic environment into a workstation is well indicated by the term: Personal University. Personal Universities have to integrate all the applications and documents of an entire university curriculum in a single workstation to be used by teachers and students. To make this possible sound architectural criteria have to be devised. This work reports reviews and suggestions taken from the literature as well from a Personal University[1] project still under development at Milan University jointly with Asunción Catholic University.

Reasons for personal university workstation

Weissman has already pointed out in 1988 [21] the significant changes that we could expect from an intensive and careful application of technology to

[1] PU is in the frame of an accademic cooperation agreement between Milan University (Italy) and Catholic University of Asunción (Paraguay). It is sponsored by the Minister of Foreign Affairs jointly with the Minister of University and Scientific Research of Italian Government.

higher education. Notebook size computers, large storage devices like CD-ROM [8,12,15], communication facilities, will provide the base for the design of new instructional systems. Nevertheless, to become largely diffused, these systems have to cope with technological and commercial constraints that prevent their advance.

The existence of different and incompatible types of components is the primary source of technological problems. Actually building an instructional system implies the selection of several elements like:

- Computer platform
- Operating system
- Graphical user interface
- Author system
- Data support and format
- Communication devices.

The result of all the possible combinations is an atomized market, influenced also by commercial problems as:

- High cost of instructional software production
- Cost of organizational efforts
- Lack of established circuits for instructional s/w distribution.

The outlined situation make almost impossible the settlement of standards (h/w, s/w, courseware) that would accelerate the growing of the sector. So, even if no discipline has been untouched by academic computing tools and technology [7,20], today tools remain far from a full integration. This becomes more evident considering a lecture, the typical interface between faculty and students. Computer support for lecture performance is still limited. Preparation of lesson is time consuming, as most of the informations needed reside on paper and have to be entered to the machine. Once the lecture is prepared its presentation to the classroom represent a technical challenge. Tuning all system components is time consuming and troublesome, also even a single h/w and s/w platform is in use. Hardware configuration, operating system version, software installation often become problems as each professor uses his own set of tools. Using a single portable workstation is a simple alternative. Provided a video signal standard, professor has just to plug in and play his own workstation in the classroom. As standards for multimedia data interchange will be defined [2, 13, 17] communication among this kind of

workstation will increase, even if they were developed using competitive technologies.

These considerations let us understand the value of making available all documents and applications for a complete university department on a single integrated environment. We call this academic integrated environment: Personal University (PU).
In a Personal University framework a professor could work out the following activities:

1. Personal time management
2. Scientific writing
3. Scientific computation
4. Personal literature
5. Access to data and literature banks
6. Cooperative work with colleagues
7. Lecture preparation
8. Drill, practice and exercise preparation
9. Student's evaluation
10. Course management
11. Communication with students
12. Lecture performance using a lecture room projector

Being higher education an intensive form of training, we assume that teachers and students have to use almost the same tool. As today they can have access to the same library, tomorrow they will have available the same information sources. A personal time management system will allow students to take care of the different subject matters that they have to master. Through PU workstation student will be able to study, he will perform evaluation of his discipline mastering. PU will be also a tool for professor during examinations.
As already outlined, an integrated environment including facilities for lectures and examinations will reduce the gap between research and study. Since teachers and students use the same environment to study, the training aspect of university life will become more evident. Teachers and students will become knowledge explorers working together. The use of PU could become an obvious activity as the use of paper.

Personal University Requirements

It is rather evident that the development from scratch of the software needed for a PU is beyond actual financial ranges of the instructional field. Indeed the budget for didactic purposes is generally limited, even if a certain number of big projects is underway[1,3]. The integration of existing application permits us to withstand economical constraints. This implies that the first mandatory requirement for PU is the *availability of operating systems that allow communication among applications*, possibly existing ones. Given the state of the art in this field, it seems correct to project the solution of a working Personal University within the next two years.

The second requirement is about *software selection*. The availability of future versions supporting interapplication communication is the primary criterion. This requirement is not stringent for developing prototypes. It will not be difficult or costly to substitute some components.

A design methodology arises naturally as third requirement: *realistic prototyping*. This methodology suggests the implementation of prototypes with the best available components (h/w or s/w), and with a minimum software development effort. The availability of communication among applications will give a large premium to this attitude when such an integration facility will be fully available. The fourth requirement (that sprang from PU experimentation) is that the *project goal has to be quite advanced.* In this way during the life time of the development cycle there will be little chance for the competition to prevail.

The fifth requirement for the development of a workstation for higher education is inherent to *publishing problems*. As we have previously observed, software distribution channels are not still adequate to distribute software instructional material. There are many reasons for this situation and it is outside our purpose to analyze them in this note.

We just want to affirm that a *good integration with paper publishers* would make things better. Juxtaposing "electronic" and hard-copy books is a suitable frame for research[22], but not for production[16,18]. Traditional editors have well established experience from various point of view regarding promotion, advertising, incentives for authors, quality control, distribution....

There are many technical routes to reach such a good integration. Here we suggest one, that could be fully sustained by paper publishers in the case that electronic media will emerge.

1 - Electronic version has to reproduce the appearance of its paper correspondent.
2 - All the snapshots of electronic version must be printable.
3 - Electronic version must be treated as an *added value* of the paper version. All the activities that can be performed by hand, with pencil and paper only, will be automated in the electronic version.

Personal University Architecture

Starting from previous requirements, only few possible choices remain. Integration with traditional publishing strongly suggests centering of Personal University on a library of books in electronic format. We call pu-book the electronic book of PU. Pu-book must satisfy the criteria pointed out in fifth requirement, and has to be well integrated with all the rest of PU to comply with all other requirements.

It easily follows that the rest of the PU has to communicate quite well with the pu-book. This will be the basis for the activities that we mentioned above both for the professors or for the students. The library, the data bank, the presentations, the communication channels, the personal documents file and all the other information sources or destinations will be integrated. Since some of these sources or destinations, elements of selected commercial applications, it follows that integration will rise from communication among applications; that is from higher order software not necessarily provided by commercial application developers. We suggest a name for this higher order software (software linking commercial applications): agents.

In our hypothesis, agents have to deal with:

- Filling up empty pu-books, that is developing or modifying a pu-book.
- Adding personal notes to the pu-book.
- Navigating the PU-library.
- Producing and performing lectures with suitable presentation systems.
- Doing computations (mathematics, graphics...) on the entities of the pu-book.

- Developing evaluation (examination) dialogues, for example multiple choice.
- Producing printed material.

...

We will not give the full list as this can grow and it is related to many factors. Obviously, standard packages suitable to make possible all the activities suggested in the first chapters have to be integrated.

The human interface: the hyper dimension

The human interface represents an important element of every computer system. There are many aspects that have to be taken into account for the choices to be done. Hypermedia is certainly a good choice for a scientific workstation [5,6,9]. Facilities to make buttons trigger other communicating applications have to be provided. For example mathematical formulas of pu-book have to be computed directly upon request to a mathematical agent. The result will be placed where it has been established during the book design, using a simple script language based on the notion of context. It is necessary to take care of similar considerations also for function graphs or other computational entities.

All reference entities are good candidates for buttons/links that can connect them in a bidirectional symmetrical way. Indexes are a collection of links to corresponding parts and all parts can be used to recover the corresponding index. The same can be carry out for figures identifiers, references, notes, formal content. Definitions will be easily referenced by a suitable agent.

To conserve the paper book structure, a default navigation path could be selected, based on the page structure. However, the reader will have other possibilities of navigation following associative links. To avoid cognitive difficulties, the path followed by the user is certainly the best entity to be used as an Arianna's wire. We call the user path *hyper dimension* or *hyper*.

Hyper is analogous to Bush concept of trail in Memex [4] and, as trail, it is essentially a medium for communication. All the information collected and the activities performed during navigation have to be stacked to hypers. The hyper dimension can obviously be stored and it is a way to build up a lecture. A lecture preparation is just a navigation in the PU-library. A lecture performance is just a reply, driven by the teacher, or by a clock, or by

students' questions, of previously prepared hyper. The information elements associated to hyper are just information taken from books, results of computations, computation requests, graphs and picture (animated or not), segment of vocal comments.

So the human interface suggests itself a style for building up lectures, which is not biased by pedagogical suggestions. Transferring traditional material into electronic form we allow an easier way of lecture construction and presentation, minimizing interferences with the teachers' educational habits. Integration between hyper dimension (an agent) and a presentation system is a natural result of this consideration, as suggested by efforts in developing the PU project. Last but not least, the hyper dimension is a good candidate for transporting work from teacher to student, from student to teacher or to other students or for cooperative learning. Hypers, just as files, can be opened, displayed, filled, closed, stored, mailed, copied, edited, combined, browsed, protected, shared....

Intelligent Tutoring in a Personal Universtiy

Navigation paths could be used by intelligent agents as a way for learner modeling [19]. Such an intelligent tutoring system will advice the user about learning strategies, retrieval of useful information, performance of practices and exercises.

Hypers are a good base for the development of a user observer as they collect all the user interactions with the system. Anyhow, to allow the use of hypers as knowledge base about learner habits, their content has to acquire a semantic significance suitable for machine interpretation.

A simple way to start intelligent tutoring is the observation of reading. A recording of the time spent on any page can provide a useful parameter to start a comparison among different learners. The sequence of browsed pages is another useful information. When a formal representation of the subject is available, a check about page sequence consistency can be performed. When some inconsistency is detected, the system could start a dialogue with the user discovering new association possibilities or advising a better sequence.

Integration with evaluation tool will offer another optimal opportunity to make the tutoring system aware of learner characteristics. Response to multiple-choice answer tests is the starting point of similar tool, but the

integration with intelligent applications will lead to more sophisticated approach. For example: intelligent tools for grammatical verification are already available. Integrating them with a pu-book would make it possible to compare detected mistakes and hyper contents, and discover where the user has to be addressed for improvements.

Formal disciplines and the pu-book

The development of pu-book has been a good chance to understand a possible use of the formal content of disciplines [11]. Indeed, given a network representing the formal description of the book subject, it becomes possible to browse through the book by a network navigation. Such a network could be based on the notion of conceptual dependency: a concept depends on others trough connections elements (nexuses). Taking concepts and nexuses together we produce a network.

Browsing in the network means stepping among concepts. For this reason we call the kind of browser we developed for the personal university, conceptual browser. Other entities can be associated to concepts and nexuses: examples, references, explanations, notes, comments, user notes, parts of text... these entities become detachable components of a formal structure of the text. In the pu-book developed by the authors Petri Nets have been used [10,14]. However any other net or graph can be used, as, for example, semantic nets.

Many suggestions emerge from the conceptual browser development. The usefulness of such a navigation for all readers is not always obvious: but if a user wants to enter deeper into the subject matter, it is a strong advantage to have the possibility of understanding its formal structure.

From the design point of view, the link between formal and informal content can be related to textual chunks producer and to maintenance facilities. Thes notions seem related to an extension of the mark-up techniques to the case where mark-up have a formally treatabl meaning.

It is probably a good line of experimentation to develop technologies that can finally show one real use of formal contents of the disciplines.

At that point theorem provers could be used for formal deductions. An intelligent dynamical book could result.

Conclusion

This paper is a mix of results and suggestions emerging from our experimental activity. The suggestions have all been tested with some prototyping activitiy. Sometimes this prototyping has been carried out in different machines, without any intention of being ready for an engineering effort, and in any case lacking a good operating system suitable for supporting communication among applications.
Results are mainly a set of simple architectural suggestions that the reader can easily find in our notes. With them we hope to contribute to minimizing mistakes in future developments.

Acknowledgements

We acknowledge HUG (Hypertext User Group) of Computer Science Department at the Universityof Milano and the Digital Lab at the University of Asunción (LED). They given a strong support to the activities. Then we give thanks to the many Italian and Paraguayan students who worked out all the prototyping. Their work will appear elsewhere and is available on request. Italian Minister of Education and the Minister for the Foreign Affairs have given financial and normative support: we deeply acknowledge them.

References

1. Balkovic E., S. Lerman., R. P. Parmelee, "Computing in Higher Education: The Athena Experience." *Computer* 18(11):112-125; November 1985.

2. Bono, P. R. "A Survey of Graphic Standards and Their Role in Information Interchange." *Computer* 18(10):15-30; October 1985.

3. Brand, S. "The Media Lab: Inventing the Future at MIT". New York: Viking Penguin Inc., 1987.

4. Bush, V. "As We May Think." *Atlantic Monthly*: 176, pp. 10-108; July 1945. Reprinted in *CD ROM: The New Papyrus*; Microsoft, 1986.

5. Conklin, J. "Hypertext: An Introduction and Survey." *Computer* 20(9):17-41; September 1987.

6. Cook, P. "Multimedia Technology." In *Interactive Multimedia.* Edited by S. Ambron and K. Hooper. Redmond, WA: Microsoft Press, 1988.

7. Degli Antoni, G. "Technology and Scientific Thinking." Atti del Congresso Internazionale SILFS. Viareggio; January 1990.

8. Elshami, A. "CD-ROM: An Annotated Bibliography." Libraries Unlimited, Inc., 1989.

9. Fraase, M. "Macintosh Hypermedia." Scott, Foresman and Co. Glenview, 1990.

10. Genrich, H. J. "The Petri Net Representation of Mathematical Knowledge." GMD Bonn, Interner Bericht, ISF-76-05, 1976.

11. Greselin, F., R. Ferrari, G. Degli Antoni, D. Viganó, "Petri Nets as 'Informal Knowledge Engineering' Solution." Proceedings XVI Conferencia Latinoamericana de Informática: 459-477; Asunción, September 1990.

12. Holtz, F., "CD-ROMs: Breackthrough in Information Storage." Tab Books, 1988.

13. Horak, W. "Office Document Architecture and Office Document Interchange Formats: Current Status of International Standardization." *Computer* 18(10):50-60; October 1985.

14. Jantzen, M. "Structured Representation of Knowledge by Petri Nets as an Aid for Teaching and Research."

15. Lyon, E. "Spoilt for Choice? Optical Discs and Online Databases in the Next Decade." Program 25(1):37-50;January 1991.

16. Myers, P. "Publishing with CD-ROM." Meckler Corporation, 1987.

17. Postel, J. B., G. G. Finn, A. R. Katz, J. K. Reynolds, "An Experimental Multimedia Mail System." ACM Transaction on Office Information Systems 6(1):63-81; January 1988.

18. Ropiequet, S., J. Einberger, B. Zoellick, "CD-ROM and Optical Pubblishing." Microsoft Press, 1987.

19. Weyer, S. A. "As We May Learn." In *Interactive Multimedia.* Edited by S. Ambron and K. Hooper. Redmond, WA: Microsoft Press, 1988.

20. Weissman, R. F. E. "Academic Computing: the Next Wave", *Brown Online* 2: 3-6; May 1989.

21. Weissman, R. F. E. "From the Personal Computer to the Scholar's Workstation", *Academic Computing* (10): 10-41; October 1988.

22. Yankelovich, N., N. Meyrowitz, A. van Dam, "Reading and Writing the Electronic Book." *Computer* 18(10):15-30; October 1985.

O Uso de Semântica na Construção e Expansão de Árvores de Decisão Indutivas

Giuseppe Mongiovi
Walfredo da Costa Cirne Filho

Universidade Federal da Paraíba
Centro de Ciências e Tecnologia
Departamento de Ciência da Computação
Av. Aprígio Veloso, s/n
Campina Grande - PB - CEP 58.100 - BRASIL
E-mail: wfredo@brufpb2.bitnet

Resumo: Neste trabalho é apresentado o problema das regras inúteis, geradas por algoritmos da família TDIDT, e uma solução baseada em semântica, que expande a árvore de decisão gerada por estes algoritmos, é analisada. É também apresentada uma nova solução para o problema, que consiste na construção direta da árvore de decisão. Esta construção é guiada por uma função de avaliação pragmática, em que os aspectos estrutural e semântico são observados. Uma análise comparativa entre as soluções é feita.

I. Introdução

Uma das principais áreas de pesquisa em Inteligência Artificial, do ponto de vista científico e comercial, constitui-se dos Sistemas Baseados em Conhecimento (em particular, dos Sistemas Especialistas). A grande dificuldade na construção desses sistemas reside no processo de aquisição do conhecimento necessário ao seu funcionamento. Para minimizar este problema foram desenvolvidos métodos automatizados para aquisição de conhecimento [Boy 87] [Boose 87]. Entre os processo automatizados, o da geração da Base de Conhecimento (BC) a partir de exemplos tem sido um dos mais explorados. Entre estes, destacam-se os da família TDIDT (*Top Down Induction of Decision Trees*) [Quinlan 86].

Esses métodos geram automaticamente uma Árvore de Decisão (AD), a partir de um conjunto de exemplos, imitando o processo humano de aprendizado por indução [Boman 89]. Isto é feito classificando os exemplos. Tal classificação é feita a partir dos pares atributo/valor que caracterizam cada exemplo. A geração é feita tentando minimizar a AD, usando uma função de avaliação (geralmente, cálculo de entropia) na escolha do melhor atributo a expandir [Mingers 89]. A AD gerada pode ser usada diretamente como BC ou transformada num conjunto equivalente de regras de produção. Entre os métodos que realizam este processo de aquisição destacam-se o ID3 [Quinlan 86], o C4 [Quinlan 87] e o CART [Breiman 84]. Estes métodos diferem entre si pela estrategia adotada para minimizar a AD.

Apesar do aparente sucesso alcançado por esses métodos, algumas limitações de cunho prático têm sido identificadas. A inabilidade de lidar com ruído nos dados de entrada, a exigência de valores discretos e disjuntos e a forma como os resultados da indução são apresentados têm sido alvo de pesquisas recentes na área. Vários trabalhos, solucionando parcialmente esses problemas, têm sido apresentados [Cendrowska 88] [Quinlan 83]. Basicamente estes trabalhos têm abordado apenas o aspecto estrutural da **AD**. Entretanto, um fato muito importante, do ponto de vista prático, não vem sendo lembrado. É o aspecto semântico da **AD** gerada. Isto é, será que todo caminho na **AD** corresponde a uma regra de utilidade prática? A resposta é não.

Este trabalho apresenta o problema das regras inúteis (regras que, embora estruturalmente corretas, não têm utilidade prática) e discute a solução apresentada pelos autores para o algoritmo **ID3** [Mongiovi 90] e posteriormente generalizada para toda a família TDIDT (algoritmo **ADEX**) [Cirne 90]. Esta solução é baseada em conhecimento e consiste na expansão semântica de uma **AD** gerada por um algoritmo TDIDT. Como, apesar da pouca validação, a expansão apresenta alguns problemas práticos, será apresentada uma nova solução para o problema das regras inúteis. Esta nova solução usa uma estratégia de indução baseada não só no aspecto estrutural mas também no aspecto semântico. Estratégia esta que representa uma grande mudança de enfoque em relação aos métodos tradicionais de indução em **AD**.

Na seção II levanta-se o problema das regras inúteis. A seção III apresenta o conceito de Matriz de Relevância (**MR**) que formaliza o conhecimento necessário para os métodos aqui descritos. A seção IV apresenta e discute o algoritmo **ADEX**. Na seção V encontram-se a descrição da nova solução proposta, denominada de algoritmo IDRT, e uma análise comparativa entre esta solução e os algoritmos **ID3** e **ADEX**. Finalmente, as conclusões estão na seção VI.

II. O problema das regras inúteis

A árvore de decisão gerada pelos algoritmos da família TDIDT é uma estrutura despida de semântica. Ela apenas mapeia, de forma reduzida, o conjunto de exemplos. Devido a este fato, estes algoritmos podem gerar, em certas circunstâncias, regras sem significado que na prática são inúteis. Uma circunstância em que isto frequentemente acontece é quando o conjunto de exemplos contém "casos raros".

Numa regra inútil, nota-se que a premissa é irrelevante para implicar a conclusão. Por exemplo, na construção de um Sistema Especialista em ginecologia, num universo de

pacientes adultas, com apenas alguns casos de crianças, o sistema de aquisição automática de conhecimento APREND [Gomes 88] [Gomes 89] forneceu a seguinte regra:

SE IDADE = criança
ENTÃO DIAGNÓSTICO = vaginite

Isto aconteceu porque no conjunto de exemplos, por coincidência, todas as crianças estavam com vaginite (já que os dados foram obtidos de fichario médico) e, portanto, o atributo **IDADE** era o de menor entropia. Além disso, o sistema APREND encerrou neste ponto a construção do ramo correspondente ao caso, já que todos os exemplos ficam classificados apenas com **IDADE = criança**. O exemplo (caso raro) ficou, portanto, mapeado numa regra inútil, uma vez que a premissa da regra acima ("a paciente é criança") é irrelevante para concluir se a paciente está ou não com vaginite. Note que, embora mais fácil de ocorrer em casos raros (pois as chances de erro numa pequena amostragem são maiores que em uma amostragem mais significativa), esse problema pode aparecer também em outros casos.

Este tipo de problema pode ocorrer até mesmo em sistemas "de brinquedo". Por exemplo, seja a tabela da figura 2.1.

	COME	DIABÉTICO	IDADE	VEGETARIANO	DIAGNÓSTICO
ex1	pouco	sim	velho	sim	MAGRO
ex2	médio	não	velho	sim	MAGRO
ex3	muito	sim	velho	não	GORDO
ex4	pouco	não	velho	não	MAGRO
ex5	médio	sim	jovem	não	GORDO
ex6	pouco	não	jovem	sim	MAGRO
ex7	muito	não	velho	não	GORDO
ex8	médio	não	jovem	não	GORDO

Figura 2.1 - Tabela de exemplos

A árvore gerada pelo sistema APREND, correspondente a tabela acima, está na figura abaixo:

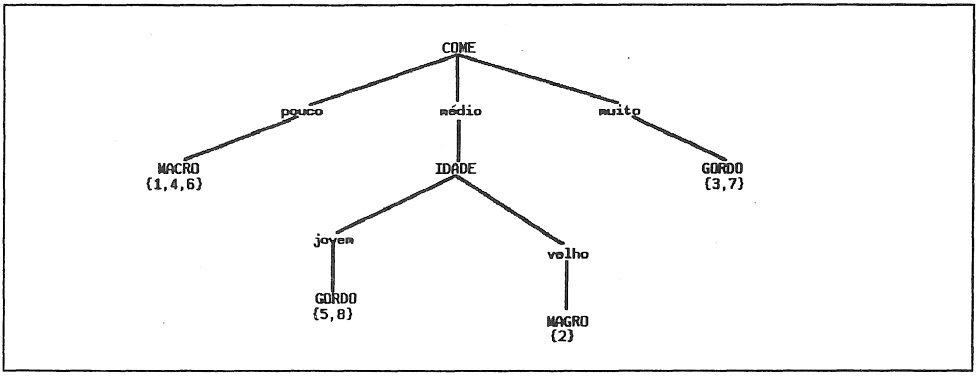


Figura 2.2 - AD gerada pelo sistema APREND

Cada folha da árvore acima é composta por um elemento de classificação e por um conjunto que representa os exemplos por ela mapeados. A informação dos nós está em maiúsculo e a dos ramos em minúsculo. As regras correspondentes à árvore são:

r_1) SE COME = pouco ENTÃO DIAGNÓSTICO = MAGRO
r_2) SE COME = médio E IDADE = jovem ENTÃO DIAGNÓSTICO = GORDO
r_3) SE COME = médio E IDADE = velho ENTÃO DIAGNÓSTICO = MAGRO
r_4) SE COME = muito ENTÃO DIAGNÓSTICO = GORDO

As regras r_2 e r_3, embora não mapeando casos raros, têm, como se pode ver, pouca utilidade prática e são ditas inúteis. Isto porque, na prática, **DIABÉTICO** e **VEGETARIANO** são fatores muito mais indicativos do que **IDADE**, para se concluir se alguém é **MAGRO** ou **GORDO**. Além do mais, o atributo **COME** não é importante para a conclusão (**MAGRO/GORDO**) com o valor **médio**.

III. Matriz de Relevância

A solução encontrada para contornar o problema das regras inúteis consiste na expansão da árvore de decisão fornecida por qualquer um dos métodos da família TDIDT. A expansão é feita a partir de uma Matriz de Relevância (**MR**), construída pelo especialista do domínio em questão.

Uma **MR** para um dado domínio é o relacionamento de relevância entre os elementos de classificação e os atributos deste domínio. Em uma **MR** a linha **i** representa o atributo A_i e a coluna **j** o elemento de classificação D_j. Um elemento R_{ij} de uma **MR** indica o grau de relevância do atributo A_i para a classificação de D_j. Este grau é dado pelo conjunto de valores

do atributo A_i que são relevantes para D_j. O elemento R_{ij} é, na verdade, um subconjunto do conjunto de valores do atributo A_i. Se R_{ij} for vazio, significa que o atributo A_i é irrelevante na classificação de D_j. Ao passo que, R_{ij} igual ao conjunto de valores de A_i significa que A_i é totalmente relevante na classificação de D_j. Uma **MR** é dita completa se todos os atributos forem totalmente relevantes para todos os elementos de classificação. Uma **MR** é vazia se todos os seus elementos forem iguais ao conjunto vazio.

Observa-se que aparentemente uma **MR** é semelhante à uma rede de repertório (*rating grid*) dos sistemas tipo ETS [Boose 84]. Entretanto, existe uma grande diferença entre elas. Enquanto numa rede de repertório o relacionamento é total (todos os elementos de classificação se relacionam com todas as características), numa **MR** apenas os relacionamentos tidos relevantes são considerados. Isto minimiza a participação do especialista no processo de aquisição de conhecimento.

Para o exemplo em questão (figura 2.1), uma **MR** poderia ser a da figura abaixo.

	MAGRO	GORDO
COME	{pouco}	{muito}
DIABÉTICO	{não}	{sim}
IDADE	Ø	Ø
VEGETARIANO	{sim}	{não}

Figura 3.1 - Exemplo de uma **MR**

A figura 3.1 mostra que:

i) O atributo **COME** é relevante para classificar **MAGRO** no valor **pouco**, e é relevante, no valor **muito**, para classificar **GORDO**.

ii) O atributo **DIABÉTICO** é relevante para classificar **GORDO** no valor **sim** e **MAGRO** no valor **não**.

iii) O atributo **IDADE** é irrelevante para classificar **MAGRO** e **GORDO** (casos deste tipo alertam ao **EC** que possivelmente o atributo **IDADE** deveria ser eliminado da tabela de exemplos).

iv) O atributo **VEGETARIANO** é relevante para classificar **GORDO** no valor **não** e **MAGRO** no valor **sim**.

IV. Algoritmo ADEX

A árvore de decisão pragmática é uma expansão da árvore gerada por um algoritmos TDIDT. A finalidade dessa expansão é colocar a semântica obtida do especialista (contida na MR) na árvore. Pretende-se, desta forma, evitar que se tenha uma regra onde as premissas são irrelevantes para a conclusão, isto é, evitar regras inúteis.

A expansão é feita de modo a colocar pelo menos uma condição relevante em todo caminho que vai da raiz a uma folha qualquer. Ou seja, toda regra terá um componente relevante em sua premissa. Abaixo, segue o algoritmo **ADEX**, que expande uma **AD** gerada por um algoritmo TDIDT em uma árvore pragmática.

```
adex(AD)
        PARA TODA folha f (composta pelo elemento de classificação D
        e pelo conjunto de exemplos E) da AD
                tentar_expandir(f);
tentar_expandir(f)
        SE nenhuma das condições pertencentes ao caminho(raiz,f) é
        relevante ENTÃO
                SEJA ai um atributo não pertencete ao caminho(raiz,f);
                SEJA Vi = {Vi1,Vi2,...,Vin} o conjunto de valores que
                são relevantes para ai concluir D;
                SEJA A = {a1,a2,...,ak} o conjunto de todos os ai's
                tais que existe um exemplo pertencente a E com o valor
                de ai pertencente a Vi.
                SE A não é vazio ENTÃO
                        SEJA am o atributo de "melhor" função de avaliação
                        pertencente a A;
                        substituir f pelo nó não-terminal am ligado pelos
                        ramos Vm1, Vm2, ..., Vmn e * às folhas f1, f2,
                        ..., fn e f*, respectivamente;
                        associar a cada uma destas folhas o elemento D e
                        redistribuir o conjunto de exemplos E entre elas;
                        tenta_expandir(f*);
                SENÃO marca f como inútil;
```

No algoritmo **ADEX**, o símbolo *, filho do atributo a_m, representa todos os valores não relevantes de a_m. Por exemplo, aplicando o algoritmo à árvore da figura 2.2, tem-se a árvore da figura 3.2.

As regras correspondentes à árvore **ADEX** são:

e_1) **SE** COME = pouco **ENTÃO** DIAGNÓSTICO = MAGRO
e_2) **SE** COME = médio E IDADE = jovem E DIABÉTCO = sim
ENTÃO DIAGNÓSTICO = GORDO
e_3) **SE** COME = médio E IDADE = jovem E VEGETARIANO = não
ENTÃO DIAGNÓSTICO = GORDO
e_4) **SE** COME = médio E IDADE = velho E DIABÉTCO = não
ENTÃO DIAGNÓSTICO = MAGRO
e_5) **SE** COME = muito **ENTÃO** DIAGNÓSTICO = GORDO

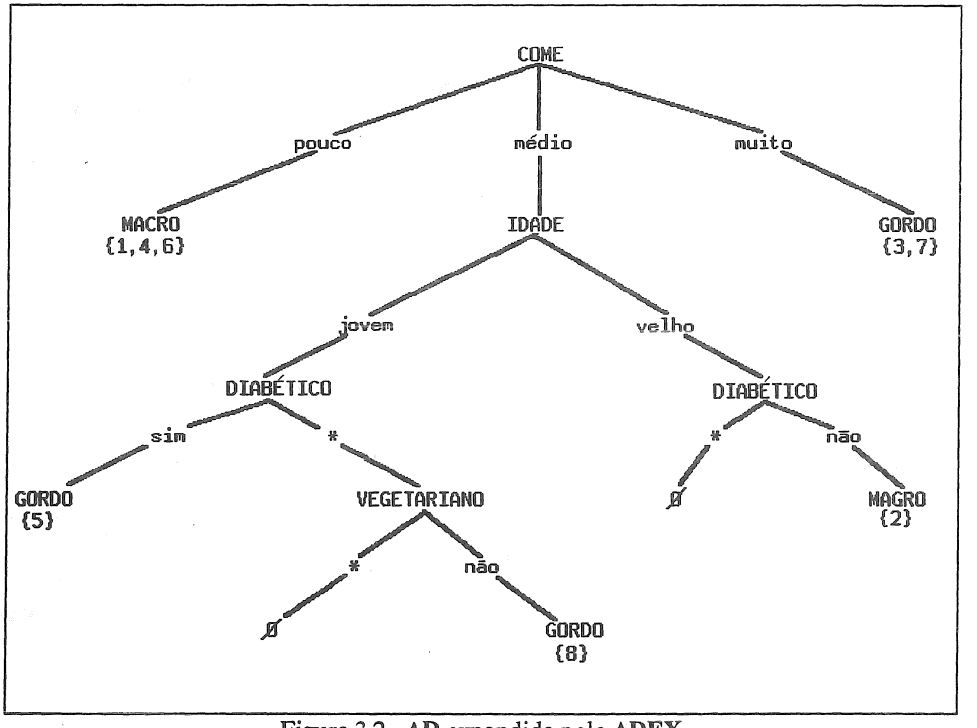


Figura 3.2 - AD expandida pelo **ADEX**

Observe que os arcos que têm como valor o * não aparecem nas regras, já que eles não acrescentam nenhuma relevância às regras originalmente obtidas. Ou seja, o símbolo * é transparente para o gerador de regras.

Da comparação das regras originais com as regras pragmáticas, nota-se que a regra r_2 inicialmente tida como inútil foi substituída, com o uso do algoritmo **ADEX**, pelas regras úteis e_2 e e_3. Além disso, a outra regra inútil r_3 foi transformada na regra útil e_4. A conclusão de que as regras e_2, e_3 e e_4 são úteis deve-se ao fato de que, na prática, os atributos **DIABÉTICO** e **VEGETARIANO**, contantes destas regras, são bastantes significativos para distinguir entre **MAGRO** e **GORDO**.

Embora no exemplo citado todas as regras inúteis foram substituidas por regras úteis, na prática isto nem sempre acontece [Cirne 90] [Mongiovi 90]. Entretanto, isso não é devido à uma limitação do **ADEX** e sim à pouca informação contida na **MR**. Mesmo quando o **ADEX** não retira regras inúteis seu uso é recompensador, pois as regras inúteis ficam marcadas de forma que quando o especialista for revisar o banco de regras obtido, pode concentrar sua atenção sobre elas.

Mesmo para os casos extremos, a **MR** completa ou vazia, o **ADEX** ainda produz resultados coerentes. Em ambos os casos ele não muda a árvore original. Se a **MR** for completa, conclui-se que todas as regras são úteis (já que tudo é relevante). Ao passo que, se a **MR** for vazia, obtem-se que todas as regras são inúteis (como nada é relevante, tudo é inútil).

V. Construção Direta de uma Árvore de Decisão Pragmática

Apesar da expansão semântica de uma árvore TDIDT melhorar a qualidade desta árvore, colocando condições relevantes nas premissas das regras, ainda restam alguns problemas. Inicialmente, nota-se que como a expansão só faz acrescentar nós à árvore, esta pode assumir proporções inadequadas. Além disso, só é garantida uma condição relevante na premissa, mesmo quando esta tem um grande número de condições.

Uma forma, sugerida pelos autores, de contornar os problemas acima mencionados consiste na construção direta da **AD** levando em consideração não só o aspecto estrutural como também o aspecto semântico. Esta construção será guiada por uma Função de Avaliação Pragmática (**FAP**) que engloba ambos os aspectos. Este novo algoritmo (que utiliza a **FAP** como função de avaliação) será chamado de **IDRT** (*Induction of Decision Relevants Trees*).

V.1. Função de Avaliação Pragmática

A **FAP** aqui proposta é definida de forma a ponderar a informação retirada do conjunto de exemplos (aspecto estrutural) com a informação obtida da **MR** (aspecto semântico). Esta ponderação será variável, permitindo ao usuário enfatizar, de acordo com a necessidade, o aspecto desejado (estrutural ou semântico). O principal fator que influencia a escolha da ponderação é o grau de confiança que se tem na **MR**. A **FAP** aqui definida implica que a construção da **AD** será feita utilizando primeiro os atributos de **FAP** mais alta, i. e., o melhor atributo a expandir será aquele de maior **FAP**. Portanto, para um dado atributo A_i, o valor da **FAP** m foi definido como:

$$m(A_i) = p \bullet N(A_i) + (1 - p) \bullet IR(A_i)$$

Onde: **p** é o fator de ponderação, sendo $0 \leq p \leq 1$

$N(A_i)$ é a função de avaliação estrutural normalizada

$IR(A_i)$ é a intensidade de relevância do atributo A_i

Na ausência de uma melhor indicação para compor uma **FAP** equilibrada, um valor razoável para **p** seria $p = 1/2$.

V.1.1. Função de Avaliação Estrutural Normalizada

$N(A_i)$ é uma função de avaliação convencional (i. e., uma função de avaliação estrutural) normalizada para o intervalo [0, 1]. A normalização deverá ser feita de forma que os "melhores" atributos obtenham os valores mais altos. Por exemplo, se a função de avaliação convencional considerada for a entropia **ent**, uma boa normalização seria:

$$N(A_i) = \exp(-ent(A_i))$$

V.1.2. Intensidade de Relevância

Para definir $IR(A_i)$, algumas definições se fazem necessárias. Considere, então:

n_v como sendo o número de valores do atributo A_i
n_d como sendo o núemro de elementos de classificação
$IR_v(v_k|A_i)$ como sendo a intensidade de relevância do valor v_k do atributo A_i

Pode-se agora definir a intensidade de relevância do atributo A_i como segue:

$$IR(A_i) = (1/n_v) \bullet \sum_{k=1}^{n_v} IR_v(v_k|A_i)$$

$$\text{Onde: } IR_v(v_k|A_i) = (1/n_d) \bullet \sum_{j=1}^{n_d} f(v_k|Dj)$$

$$\text{Sendo: } f(v_k|Dj) = \begin{cases} 1, \text{ se } v_k \text{ pertence a } Dj \\ 0, \text{ caso contrário} \end{cases}$$

V.2. Exemplo de uma AD IDRT

Considerando os mesmos dados de entrada (conjunto de exemplos da fig. 2.1 e **MR** da fig. 3.1) utilizados no exemplo com o **ADEX**, o **IDRT** gera a **AD** mostrada na figura abaixo.

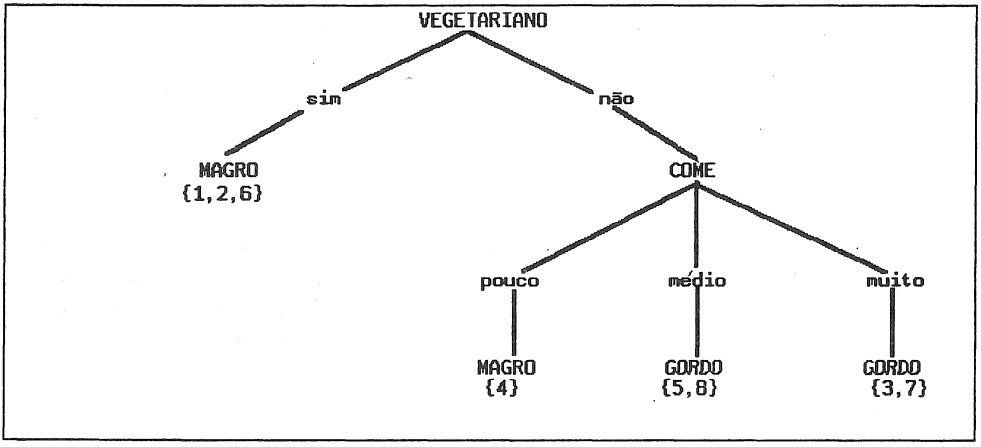


Figura 5.1 - **AD** gerada pelo algoritmo **IDRT**

A **AD** da figura 5.1, por sua vez, pode ser mapeada no seguinte conjunto de regras:

p_1) **SE** VEGETARIANO = sim **ENTÃO** DIAGNÓSTICO = MAGRO
p_2) **SE** VEGETARIANO = não **E** COME = pouco **ENTÃO** DIAGNÓSTICO = MAGRO
p_3) **SE** VEGETARIANO = não **E** COME = médio **ENTÃO** DIAGNÓSTICO = GORDO
p_4) **SE** VEGETARIANO = não **E** COME = muito **ENTÃO** DIAGNÓSTICO = GORDO

Analisando as regras fornecidas pelo **IDRT** e comparando-as com as do **ID3** e do **ADEX**, conclui-se que as primeiras são melhores que as duas últimas, visto que:

i) As regras do **IDRT**, apesar de ligeiramente maiores que as do **ID3**, em sua totalidade, são úteis do ponto de vista semântico.

ii) Embora as regras do **ADEX** sejam todas úteis, semanticamente falando, elas são bem maiores que as do **IDRT**.

Apesar dos bons resultados obtidos neste exemplo, não se pode afirmar *a priori* que isto ocorerá em todos os casos. No entanto, pelas idéias qui apresentadas, espera-se que este resultado se repita para a maioria dos casos.

VI. Conclusão

Sendo a aquisição de conhecimento o ponto de estrangulação no desenvolvimento de sistemas baseados em conhecimento, todo esforço no sentido de se obter novos métodos de aquisição ou melhorar os métodos existentes é de suma importância. O processo de aquisição automática de conhecimento a partir de exemplos é um dos mais promissores e, por isto, vem sendo alvo de frequentes investigações.

Neste trabalho são apresentadas, analisadas e comparadas duas soluções para o problema das regras inúteis. A primeira (**ADEX**), originalmente encontrada para o **ID3**, e posteriormente generalizada para todos os algoritmos da família TDIDT, consiste em expandir semanticamente as **AD**'s geradas com uso de conhecimento (**MR**) fornecido pelo especialista. A segunda (**IDRT**), uma nova solução proposta pelos autores, consiste na construção direta da árvore de decisão, onde esta construção é guiada por uma função de avaliação pragmática que engloba não só o aspecto estrutural como também o aspecto semântico.

Numa análise comparativa, um mesmo exemplo foi utilizado pelos três algoritmos (**ID3**, **ADEX** e **IDRT**). Observando-se as regras geradas pelo **IDRT** nota-se que:

i) Do ponto de vista semântico elas são equivalentes às regras obtidas pelo **ADEX** e bem superiores às do **ID3**.

ii) Do ponto de vista estrutural (quantidade de regras mais comprimento das premissas) as regras **IDRT** são menores que as do **ADEX** e praticamente iguais as do **ID3**.

Esse resultado favorável do **IDRT** é explicado pelo fato de que na construção da **AD**, o algoritmo considera não só o aspecto estrutural mas também o aspecto semântico. Embora o resultado apresentado pelo **IDRT** seja encorajador, os autores estão cientes de que isto poderá nem sempre ocorrer. Uma análise da estabilidade do **IDRT** e ajustes na função de avaliação pragmática se fazem necessários.

Referências

[Boman 89] Boman, M.; *The Problem of Induction*; SYSLAB working paper nº 152; University of Stockholm; 1989.

[Breiman 84] Breiman, L., Friedman, J., Olshen, R. e Stone, C.; *Classification and Regression Trees*; Wadsworth; Belmont; 1984.

[Boose 84] Boose, J. H.; *Personal Construct Theory and the Transfer of Human Expertise*; Anais do AAAI; 1984.

[Boose 87] Boose, J. H.; *Expertise Transfer and Complex Problems: Usign AQUINAS as a Knowledge*; Aquisition Workbench for Knowledge-Based Systems; II Man-Machine Studies; Academis Press; London; 1987.

[Boy 87] Boy, G., Faller, B. e Sallantin, J.; *Acquisition et Ratification de Connaissances*; Relatório Técnico do Centre de Recherche en Informatique de Montpelier; CRIM; 1987.

[Cendrowska 88] Cendrowska, J.; *PRISM: An Algorithm for Inducing Modular Rules*; in Knowledge-Based Systems, vol. 1; Academic Press; 1988.

[Cirne 90] Cirne Filho, W. C. e Mongiovi, G.; *Expansão Semântica para os Métodos de Aquisição de Conhecimento da Família TDIDT*; Anais do 1º Simposio de Inteligencia Artificial y Robotica; Luján - Buenos Aires - Argentina; 1990

[Gomes 88] Gomes, F. A. C.; Mongiovi, G.; Silva, H. M.; *APREND - Um Sistema de Aquisição Automática de Conhecimento a Partir de Exemplos*; 14ª Conferência Latinoamericana de Informática; Buenos Aires; Outubro/1989.

[Gomes 89] Gomes, F. A. C.; *APREND: Um Sistema de Aquisição Automática de Conhecimento a Partir de Exemplos*; Dissertação de Mestrado; Mestrado de Informática da UFPB; Campina Grande; 1989.

[Mingers 89] Mingers, J.; *An Empirical Comparison of Selection Measures for Decision-Tree Induction*; Machine Learning 3; pg. 319-342; 1989.

[Mongiovi 90] Mongiovi, G. e Cirne Filho, W. C.; *Um Algoritmo Baseado em Conhecimento para Ampliar a Potencialidade do ID3*; Anais do 7º Simpósio Brasileiro de Inteligência Artificial; Campina Grande; 1990.

[Quinlan 83] Quinlan, J. R.; *Learning from Noisy Data*; Proceedings of the International Machine Learning Workshop; University of Illinois; 1983.

[Quinlan 86] Quinlan, J. R.; *Induction of Decision Tree*; in Machine Learninig Journal I; Kluver Academic Publisher; 1986.

[Quinlan 87] Quinlan, J. R.; *Inductive Knowledge Aquisition: A Case Study*; in Application of Expert Systems; Addison-Wesley; 1987.

IV CONFERENCIA VENEZOLANA
DE INTELIGENCIA ARTIFICIAL
Y SISTEMAS EXPERTOS

CARACAS, JULIO 1991

DIANA
SISTEMA EXPERTO PARA EL DIAGNOSTICO DE FALLAS EN MINIPLANTAS COMPRESORAS DE GAS

Rafael Ramírez, Daniel Martínez, Luis Lugo, Mike Backhurst.

PALABRAS CLAVES:
Sistema Experto
Diagnóstico de Fallas
Inteligencia Artificial Aplicada
Miniplantas de Compresión de Gas
Sistema Experto Aplicado a Mantenimiento

RESUMEN

DIANA es un sistema experto que tiene como objetivo apoyar al personal de mantenimiento y operación de las miniplantas en la labor de diagnóstico y corrección de fallas. El sistema proporciona asesoría experta en relación con las fallas asociadas a 63 alarmas, secuencia de arranque y desviaciones en el comportamiento esperado de los equipos mayores.

En Lagoven S.A. existen en la actualidad 13 miniplantas, distribuidas en el lago de Maracaibo. Estas miniplantas, junto con las plantas de conservación, son las encargadas de comprimir el gas que se obtiene del crudo extraido de los yacimientos. Este gas, una vez comprimido, es liberado para su uso como gas de levantamiento artificial, para la generación de electricidad, y como gas combustible en las refinerías y en las plantas petroquímicas.

Con la incorporación de *DIANA* se espera optimar la labor de mantenimiento disminuyendo el tiempo requerido para efectuar un diagnóstico preciso de las fallas, reduciendo los tiempos de paro y disminuyendo el número de arranques necesarios para reestablecer las condiciones operativas de la miniplanta despues de una falla en las mismas; todo esto enmarcado dentro de una política de mejoramiento constante de la calidad del trabajo.

I. INTRODUCCION

Desde tiempos remotos el hombre ha venido creando herramientas cada vez más poderosas para multiplicar sus habilidades y sus limitadas capacidades físicas; sin embargo, las herramientas de trabajo desarrolladas hasta hace poco siempre se habían caracterizado por seguir el mismo patrón de apoyo al hombre en la realización de tareas que requieren cierto esfuerzo físico o habilidad manual. En estos últimos años, sin embargo, se ha dado un salto tecnológico muy significativo con el desarrollo y la comercialización de productos basados en la Inteligencia Artificial. Entre estos productos, los que mayor impacto han causado a nivel de reducción de costos e incremento de productividad son los denominados sistemas expertos o sistemas basados en conocimiento (*Knowledge Based Systems*).

Con el surgimiento de la tecnología de los sistemas expertos se ha logrado poner al alcance de la industria y del comercio, herramientas que en lugar de automatizar el esfuerzo físico, buscan multiplicar la capacidad de razonamiento del hombre, preservando en medios automatizados el recurso más valioso con que cuenta toda organización: la experiencia y el conocimiento del negocio.

En Lagoven S.A., conscientes de la importancia estratégica que reviste esta tecnología se ha dado inicio a algunos proyectos en esta área. Entre estos proyectos cabe destacar el desarrollo de *DIANA* (*DIAgNóstico Automatizado*), un sistema experto para el diagnóstico de fallas en miniplantas de compresión de gas.

El presente trabajo resume los aspectos técnicos y metodológicos relacionados con el desarrollo de esta aplicación, la aplicabilidad de esta tecnología en nuestras operaciones y la experiencia ganada en la organización de un proyecto de esta naturaleza.

II. FUNCION DE LAS MINIPLANTAS EN EL CONTEXTO OPERATIVO DE LAGOVEN

La función básica de esta empresa es la producción de crudo, tanto para la exportación como para el consumo interno. La capacidad instalada de producción que posee actualmente Lagoven representa el 45% de la capacidad instalada nacional, lo que la sitúa como la primera productora de petróleo del país. Dentro de este contexto, las miniplantas son las responsables del 25% de la producción.

Para entender mejor el rol que juegan las miniplantas dentro del proceso productivo Lagoven, es necesario revisar algunos aspectos técnicos relacionados con la producción de crudo y gas.

El crudo extraído de los yacimientos es enviado a las estaciones de flujo, en donde se somete a diversos procesos para separar el gas asociado. Este gas (el cual es una mezcla de gas asociado y gas reciclado usado en la producción por "gas-lift"), es enviado, a través de un vasto sistema de líneas, a las plantas y miniplantas de conservación, en donde se efectúa la compresión y la extracción de LPG (gas líquido del petróleo).

Lagoven posee tres tipos de plantas de compresión de gas:

- Las viejas plantas de conservación, las cuales usan una serie de compresores, accionada por una turbina industrial a gas. Estas plantas tienen una alta capacidad de compresión y las mismas tienen personal asignado en sitio las 24 horas del día. Actualmente existen cuatro de estas plantas en el lago.

- Las nuevas plantas modulares, las cuales tienen dos o más unidades con sus compresores montados en línea junto con la turbina y el generador de gas. Estas plantas tienen personal asignado permanentemente en sitio y su capacidad de compresión es inferior al de las grandes plantas. Lagoven dispone de tres de estas plantas en la actualidad.

- Por último tenemos las miniplantas, las cuales poseen dos o tres compresores montados en línea en conjunto con la turbina de potencia y el generador de gas, el cual es un derivado de aviación. Estas miniplantas tienen una capacidad de compresión inferior a las plantas modulares, sin embargo el número de miniplantas instaladas es superior a los tipos de plantas anteriores (13 en total) y las mismas no poseen personal asignado permanentemente en sitio.
  La importancia de estas miniplantas radica en que el gas que comprimen es usado fundamentalmente para la recuperación de crudo mediante levantamiento artificial "gas-lift", de allí que las mismas deban operar continuamente, excepto cuando haya algún paro programado.

III. DESCRIPCION DE LAS MINIPLANTAS

Las miniplantas están constituidas por un conjunto de sistemas entre los cuales podemos mencionar:

a.- Sistema Motor Principal.
b.- Compresores.
c.- Sistema de Gas Combustible.
d.- Drenaje de Condensados.
e.- Agua de Enfriamiento.
f.- Aire de Instrumentos y Servicios.
g.- Control Anti-Surge.
h.- Sistema de Lubricación.
i.- Sistema de Aceite de Sello.
j.- Control de Velocidad, etc.

El sistema motor principal es el corazón de la miniplanta . Este sistema le proporciona la energía necesaria a los compresores para que puedan realizar su trabajo de compresion del gas natural (ver Figura 1). El mismo está constituído por una cámara de succión, un generador a gas (turbina de acción) y una turbina de potencia (turbina de reacción).

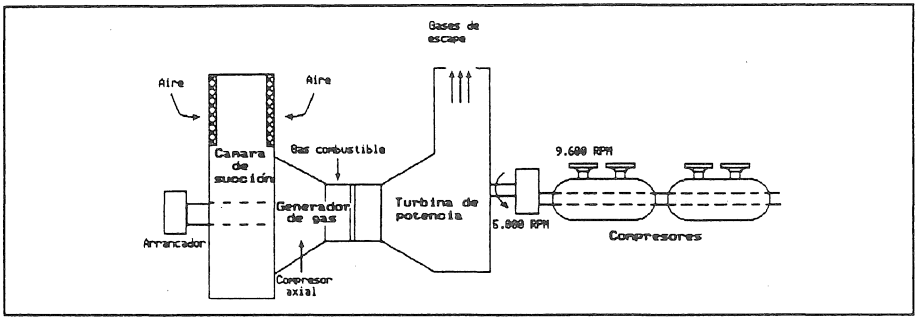


Figura 1. Sistema Motor de la Miniplanta

La cámara toma el aire del medio ambiente y lo filtra. Este aire limpio es succionado posteriormente por el generador a gas. Aquí en el generador, el cual es un derivado de aviación (jet), se hace pasar el aire por un compresor axial, se mezcla con gas combustible y se quema en los combustores. La fuerza expansiva del gas caliente (1.150 ºF) producido por el jet hace girar la turbina de potencia a una velocidad promedio de 5.000 RPM. Finalmente, la turbina de potencia está acoplada al tren de compresores mediante una caja de engranajes, la cual multiplica la velocidad proporcionada por la turbina de potencia llevándola a 9.600 RPM aproximadamente.

Al presentarse una condición operacional que ponga en peligro los equipos mayores, fundamentalmente los equipos del sistema motor principal, inmediatamente los sistemas de vigilancia inician la secuencia de paro de la miniplanta. Hacia tierra es enviada una señal por el sistema de telemetría indicando que ha ocurrido un paro en determinada miniplanta, sin especificar las razones del mismo. En el panel de alarmas del cuarto de control queda encendida (destellando) la luz indicadora de la alarma que causó el paro.

IV. DIAGNOSTICO Y CORRECCION DE FALLAS

En las miniplantas no existe personal permanente en el sitio. El trabajo de mantenimiento es realizado por tres cuadrillas a cada una de las cuales se les asigna determinadas miniplantas para su inspección. Estas cuadrillas están conformadas por un electricista, un instrumentista y un operador. La complejidad de los sistemas existentes en la miniplanta y las múltiples interrelaciones entre los mismos, hace que la labor de diagnóstico exija un amplio conocimiento y experiencia en la miniplanta para que la misma sea efectiva. Así, por ejemplo, una falla causada por la alarma de "sobrevelocidad de la turbina de potencia paro electrónico" puede ser ocasionada por instrumentos de control (caja EGM, válvula Woodward, SPEC200, monitor

de velocidad), problemas mecánicos (pickup magnéticos, mecanismo de sobrevelocidad, válvula mariposa), problemas eléctricos (relés 12TX y 12T1X, problemas en el cableado), problemas electrónicos (tarjeta del panel de alarmas), etc. Si consideramos que en la miniplanta existen 63 alarmas diferentes, podremos tener una idea de la variedad del conocimiento requerido en el diagnóstico de fallas. Ahora bien, mucho de este conocimiento es de caracter heurístico: reglas del buen juicio adquiridas con los muchos años de trabajo ininterrumpido en las miniplantas. El resto es información que está dispersa en diferentes manuales técnicos y planos, tanto eléctricos como de instalación, los cuales usualmente no están a la disposición del personal de mantenimiento en la miniplanta.

Por otro lado, aparte del posible impacto en la producción de crudo, una falla repetitiva en la miniplanta trae como consecuencia que sea necesario efectuar varios arranques de la misma. Cada arranque de estos impone sobre los equipos mayores, especialmente el generador de gas y la turbina de potencia, una sobrecarga por choque térmico, lo cual hace que disminuya su vida útil. La primera etapa de la turbina de potencia, por ejemplo, tiene una vida útil estimada equivalente a unos 500 arranques. Después de este período es necesario efectuar reparaciones mayores en la misma lo cual resulta bastante costoso debido al reemplazo de componentes importados y al tiempo de paro de la miniplanta.

V. GENESIS DE DIANA

A finales del año 1988, la organización de Computación y Sistemas de Lagoven decide llevar adelante un proyecto de desarrollo de un prototipo en el area de sistemas expertos, con el objeto de evaluar la factibilidad de aplicación de este tipo de tecnología en las operaciones de la empresa. Después de efectuar una serie de entrevistas a los supervisores de las diferentes áreas operacionales de la empresa, se lograron detectar varias aplicaciones potenciales. Entre todas estas surgen las miniplantas como el área idonea para iniciar un proyecto piloto de sistema experto.

Las razones para la selección de las miniplantas para el desarrollo de un sistema experto fueron las siguientes:

1. La tarea de diagnóstico es realizada por un equipo de trabajo multidisciplinario. Esta situación dificultaba el desarrollo de un procedimiento estándar para el diagnóstico de fallas.
2. La pericia se requiere en diversos sitios. Existen 13 miniplantas localizadas en diferentes áreas del lago.
3. Las miniplantas operan las 24 horas del día.
4. La experiencia promedio del personal que labora directamente en los recorridos de inspección de las miniplantas es relativamente bajo (3.5 años). Se dispone de personal con vasta experiencia; sin embargo, su número es insuficiente para cubrir las necesidades existentes.

5. Existe un alto beneficio económico en la reducción del tiempo de paro de las miniplantas. Aparte de los ahorros generados por el mantenimiento de los niveles de producción programados, la reducción de los costos de mántenimiento por sí sola justifica el costo de incorporar nuevas tecnologías que alarguen la vida util del equipo rotativo instalado en las miniplantas. El reemplazo total de una turbina de potencia le cuesta a Lagoven un monto en el orden de 62.5 MMBs. Una reparación mayor puede costar entre 12.5 y 42.5 MMBS.

Por todas estas razones se decidió llevar adelante el proyecto de un sistema experto que le ayudara al personal de mantenimiento a óptimar la labor de diagnóstico y corrección de fallas. Para éllo se desarrolló inicialmente un prototipo del sistema.

VI. DESARROLLO DEL PROTOTIPO

Como alcance del prototipo se seleccionaron las alarmas correspondientes al control de velocidad de la miniplanta (8 alarmas en total), lo cual era una muestra representativa del conjunto de alarmas. Para el desarrollo del prototipo se utilizó un "shell" (software para el desarrollo de sistemas expertos) sencillo y de bajo costo[1].

Previo al inicio del proyecto se conformó un equipo de trabajo integrado por dos ingenieros del conocimiento y tres expertos pertenecientes a las organizaciones de Mantenimiento y Construcción, Servicios Eléctricos y Plantas de Conservación respectivamente. De esta manera se integró en el proyecto a un representante de cada una de las organizaciones involucradas en la inspección y el mantenimiento de las miniplantas.

A medida que se obtuvieron los primeros resultados con la codificación de la base del conocimiento se pudo detectar la necesidad de soportar el conocimiento del sistema con información de tipo gráfico. De esta manera el usuario estaría en capacidad de reconocer inmediatamente los equipos a revisar y determinar su exacta ubicación dentro de la miniplanta, agilizando así la labor de diagnóstico.

Durante el desarrollo del prototipo se utilizó un microcomputador portátil. Esto facilitó implantar una forma de trabajo en la cual el conocimiento aportado por los expertos en una determinada sesión de trabajo ya estaba disponible en el microcomputador para la siguiente sesión. La interacción inmediata de los expertos con el prototipo permitió obtener un "feedback" oportuno y mejorar el conocimiento aportado en las sesiones anteriores.
Este método de trabajo, el cual se conoce con el nombre de "rapid prototyping" , ayudó a mantener el interés y la motivación de los expertos, elemento de suma importancia para el éxito de un proyecto de esta naturaleza.

Como fuente para el desarrollo de los gráficos se usaron los planos eléctricos y los diagramas esquemáticos existentes en los manuales de los equipos.

Una vez completado el desarrollo del prototipo se hizo una serie de presentaciones del mismo a los niveles supervisorios con el objeto de divulgar la tecnología y obtener aprobación para continuar con el desarrollo del sistema.

VII. DESARROLLO DEL SISTEMA EXPERTO

Para el desarrollo del sistema se seleccionó una herramienta con capacidad para representar el conocimiento mediante reglas y encuadres (frames) y con facilidades para incorporar gráficos e imágenes activas.

A. Metodología de Trabajo

En esta fase del proyecto se continuó usando la metodología de "rapid prototyping", con la variante de efectuar una revisión más completa de los árboles de inferencias antes de codificarlos en el sistema. El análisis de cada alarma se iniciaba haciendo una tormenta de ideas, a objeto de determinar todos los posibles orígenes de la falla en cuestión. Estos orígenes o causas de la falla eran posteriormente clasificados considerando aspectos tales como:

1. Probabilidad de ocurrencia de la falla.
2. Tiempo requerido para efectuar la inspección del equipo.
3. Posibilidad de hacer inspección visual.
4. Necesidad de utilizar equipo sofisticado para el diagnóstico.

De esta manera se organizó, para cada alarma, la secuencia óptima de verificación de los equipos, aquella que permitiera detectar el origen de las fallas con un mínimo de costo y tiempo.

Uno de los primeros hechos que se trata de determinar en cada una de las alarmas es si la misma fue real o fue provocada por una falla en los sistemas de vigilancia de la miniplanta. Así, por ejemplo, al iniciar una consulta de la alarma "Sobrevelocidad de la Turbina de Potencia", el sistema le pide al operador que revise, en el cuarto de control, el registrador de la velocidad de la turbina, a fin de determinar si hubo alguna desviación de la velocidad por encima de lo normal. Para éllo el sistema le proporciona al operador una velocidad de referencia.

Una vez desarrollados y revisados los árboles de inferencia, se procedió a evaluar los requerimientos de información gráfica. A diferencia del prototipo, en el cual se utilizaron los manuales de los equipos para obtener los gráficos, durante el desarrollo del sistema experto se recopiló directamente esta información en las miniplantas. Para ello se tomaron fotografías de los equipos e instrumentos siendo estos desarrollados e incorporados posteriormente en el sistema haciendo uso de un paquete de software de graficación[2].

B. Incorporación de Funciones Ad-Hoc

Muchos de los equipos de la miniplanta, especialmente los instrumentos de control requieren, para su verificación, de un procedimiento detallado de chequeo y calibración. Así por ejemplo, la verificación de la caja EGM, la cual es un instrumento que toma parte en el control de velocidad de las turbinas, requiere

de un procedimiento de calibración de unos doce pasos, los cuales, en muchos casos, es necesario repetir hasta lograr que el instrumento funcione en la forma esperada. Si esta información estuviera incorporada directamente en las reglas del sistema la base del conocimiento se hubiese complicado debido a la necesidad de incorporar lazos de control. Por otro lado, el usuario hubiese tenido que interactuar mucho más con el sistema, al efectuar procedimientos de calibración. La solución a este problema consistió en incorporar a la base de conocimientos una función o subrutina en LISP (Lenguaje de Programación), la cual se encarga de mostrar la secuencia de gráficos asociada con los procedimientos. De esta manera el usuario puede ir hacia adelante o hacia atrás en el procedimiento según su necesidad particular de información, sin obligarlo a interactuar muchas veces con el sistema.

El "shell" utilizado permite incorporar a la base de conocimientos funciones desarrolladas en casa, aumentando de esta manera la potencialidad y flexibilidad del sistema[3]-[4].

VIII. ESTRUCTURA DE LA BASE DEL CONOCIMIENTO

El sistema *DIANA* está conformado por tres grandes módulos:

1.- Alarmas de paro y auxiliares,
2.- Secuencia de arranque de la miniplanta y,
3.- Evaluación de parámetros de operación.

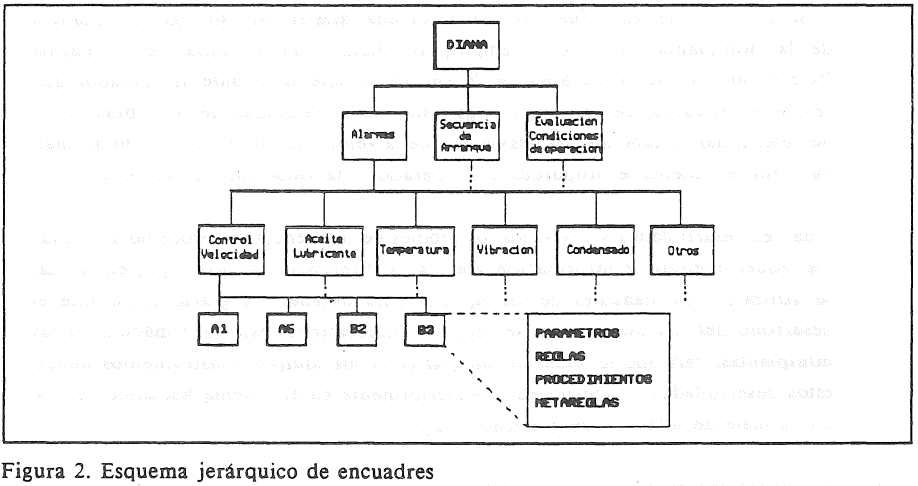


Figura 2. Esquema jerárquico de encuadres

El Módulo de Alarmas tiene por objeto apoyar el diagnóstico una vez que la miniplanta se ha parado debido a una falla o cuando existe alguna anormalidad evidente que debe ser atendida, sin que por ello se haya llegado a parar la miniplanta. La secuencia de arranque brinda apoyo en el diagnóstico de las fallas que se pueden presentar durante las diferentes etapas por las que tiene que pasar la miniplanta desde que se pulsa el botón de arranque hasta

que empieza a comprimir gas. La evaluación de parámetros de operación permite hacer un análisis, en cualquier momento, del comportamiento de los parámetros para determinar posibles desviaciones o degradaciones del comportamiento esperado de los equipos durante la operación normal de la miniplanta.

El conocimiento en el sistema experto ha sido estructurado mediante encuadres (frames), reglas de producción, parámetros y meta-reglas.

A. **Encuadres**

Los encuadres permiten dividir y estructurar la base del conocimiento en forma jerárquica, especie de módulos de programas. En *DIANA*, a cada una de las alarmas se le asoció un encuadre particular. Posteriormente todas aquellas alarmas relacionadas entre sí fueron agrupadas a su vez dentro de un encuadre común, así, por ejemplo, todas las alarmas asociadas con velocidad fueron agrupadas bajo un encuadre común denominado "velocidad" (Ver Figura 2). De igual manera se hizo con las alarmas de temperatura, vibración, aceite lubricante, etc. Esta organización del sistema permitió definir parámetros comunes a diferentes encuadres una sola vez, reduciendo de esta manera la posibilidad de inconsistencias en la base del conocimiento.

B. **Parámetros**

Los parámetros, por su parte, son estructuras que identifican o contienen trozos de información que el sistema experto usa para llegar a una conclusión. Así, por ejemplo, la PRESION_DE_SUCCION es uno de los parámetros usado por *DIANA* para tratar de determinar las causas de una sobrevelocidad en la turbina de potencia.

C. **Reglas de Producción**

Las reglas de producción, o simplemente reglas como también se les conoce, son lo que podriamos denominar las moléculas del conocimiento. Cada regla en la base de conocimiento consiste de una premisa (IF) y de una conclusión (THEN). La parte "IF" lista una serie ordenada de condiciones asociadas con los diferentes parámetros o elementos de la miniplanta que deben ser verificadas por el usuario. La parte "THEN" tiene las conclusiones y recomendaciones que especifican las tareas que debe emprender el usuario para corregir la falla que presenta la miniplanta.

La base de conocimientos de DIANA contiene alrededor de 1400 reglas de producción en estos momentos.

D. **Meta-reglas**

Por último, las meta-reglas son reglas que permiten incorporar cierta información de control para optimizar la ejecución del sistema. Las mismas son de particular utilidad en el caso de tener grandes bases de conocimiento y sea necesario mejor

la eficiencia de las mismas. En el caso de *DIANA*, las mismas fueron utilizadas basicamente para ejecutar las funciones desarrolladas en lenguaje LISP que muestran los procedimientos de verificación de los equipos.

IX. INTERFASE CON EL USUARIO

La interacción del usuario con el sistema es muy amigable. Para lograr esto se le incorporó al sistema una gran cantidad de gráficos a color con los equipos de la miniplanta. Así, por ejemplo, al empezar un sesión de trabajo, el sistema le muestra al operador el gráfico del panel de alarmas que existe en el cuarto de control. El usuario puede en este momento seleccionar la alarma con la que desea trabajar colocando el cursor sobre la alarma correspondiente en dicho gráfico.

El uso de estos gráficos ha demostrado ser de gran utilidad, especialmente cuando los equipos a verificar son difíciles de ubicar dentro de la miniplanta o cuando la verificación se tenga que hacer sobre algún dispositivo interno de los equipos. En este último caso los gráficos tipo "corte" facilitaron mucho la explicación del procedimiento a seguir.

Durante la interacción con *DIANA*, el usuario está en capacidad de modificar la línea de razonamiento para probar diferentes alternativas en un momento dado sin necesidad de suministrarle nuevamente todos los parámetros al sistema. Para éllo el usuario solamente tiene que variar aquellos parámetros que desea probar agilizando de esta manera el proceso de consulta al sistema.

Por último, los mensajes del sistema fueron traducidos del inglés al español, para facilitar el uso de *DIANA* por parte del personal de mantenimiento de las miniplantas.

X. IMPLANTACION DEL SISTEMA EXPERTO

Para la implantación del sistema se evaluaron las siguientes alternativas:

1.- Microcomputador en tierra.
2.- Microcomputador portátil.
3.- Microcomputador en la miniplanta.

La primera alternativa implica la instalación de una microcomputadora convencional en las oficinas de los supervisores. En caso de una falla en la miniplanta se establece contacto por radio con el supervisor de guardia quien desde tierra guía el diagnóstico haciendo uso del sistema.

En la segunda alternativa se le asigna a cada cuadrilla una microcomputadora portátil con el sistema experto. En este caso, el personal tiene acceso directo al sistema.

La tercera alternativa implica la instalación de un microcomputador industrial en cada una de las miniplantas. Estos microcomputadores industriales son máquinas compatibles con los microcomputadores personales pero diseñados para operar en condiciones ambientales extremas.

La estrategia de implantación finalmente acordada consiste en la implantación del sistema en dos fases. En una primera fase se ha instalado el sistema en tierra. De esta manera el supervisor de guardia dispone de una herramienta de apoyo cuando así lo requiera. Además el personal de mantenimiento se irá entrenando en el uso del sistema. Para este año esta prevista la completación de la segunda fase, la cual contempla la instalación del sistema en micros industriales en las miniplantas y el desarrollo de un módulo de diagnóstico en tiempo real en tierra acoplado al sistema SCADA. El módulo "standalone" de DIANA continuará apoyando al personal de mantenimiento en el diagnóstico de aquellos equipos que no estén acoplados al sistema SCADA, mientras que el módulo en tiempo real permitirá evaluar el comportamiento en el tiempo de los parametros de operación de la miniplanta, con el objeto de detectar en forma oportuna desviaciones en el comportamiento de los equipos mayores: compresores, turbina de potencia y generador de gas.

XI. BENEFICIOS DEL SISTEMA

Entre los beneficios esperados de *DIANA* tenemos:

1.- Reducción del tiempo de paro y de los arranques de las miniplantas.
2.- Disminución de las reparaciones de los equipos mayores.
3.- Mayor flexibilidad del sistema de gas lift.
4.- Mantenimiento de la experticia en la organización.
5.- Entrenamiento acelerado del nuevo personal.
6.- Mejor uso del tiempo de los expertos.
7.- Mantenimiento de una historia confiable de fallas.
8.- Mejoramiento constante en la calidad del trabajo.

XII. CONCLUSIONES

1. El uso de un sistema experto en áreas que requieren un conocimiento multidisciplinario es de suma utilidad para estandarizar los procedimientos de diagnóstico y corrección de fallas.

2. La metodología de "rapid prototyping" o desarrollo incremental es muy apropiada para facilitar la comunicación con los expertos y para obtener un feedback oportuno de los mismos. Esto ayuda además a mantener la motivación de los expertos y a lograr una participación más activa de los mismos en el desarrollo del sistema.

XIII. RECOMENDACIONES

1. antes de iniciar un proyecto mayor en el área de sistemas expertos, es conveniente desarrollar un prototipo para determinar la factibilidad y los alcances del mismo y cual es el hardware y el software más apropiados para el desarrollo del sistema.

2. Para el desarrollo del prototipo es recomendable usar un shell sencillo y de bajo costo especialmente cuando se vaya a desarrollar un primer sistema experto en la organización y no exista experiencia previa en la materia.

3. Es recomendable diseñar una interfase de usuario lo más amigable posible; preferiblemente de tipo gráfico.

4. Antes de iniciar un proyecto de sistema experto es conveniente hacer un estudio para determinar las areas potenciales con mayor rentabilidad y probabilidad de éxito en la aplicación de esta tecnología dentro de la empresa.

5. Es conveniente utilizar herramientas que permitan la integración a los sistemas de información existentes y la conexión a sistemas de adquisición de datos en tiempo real.

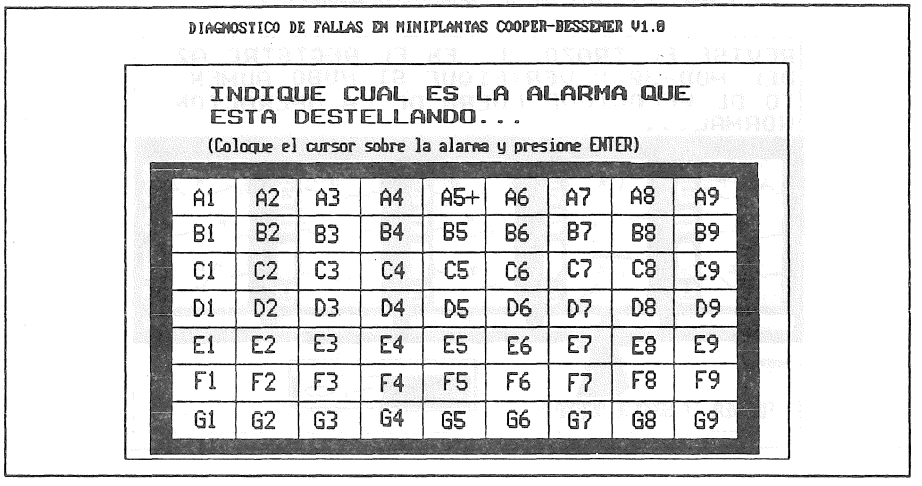


Figura 3
Inicio de una consulta en *DIANA*.

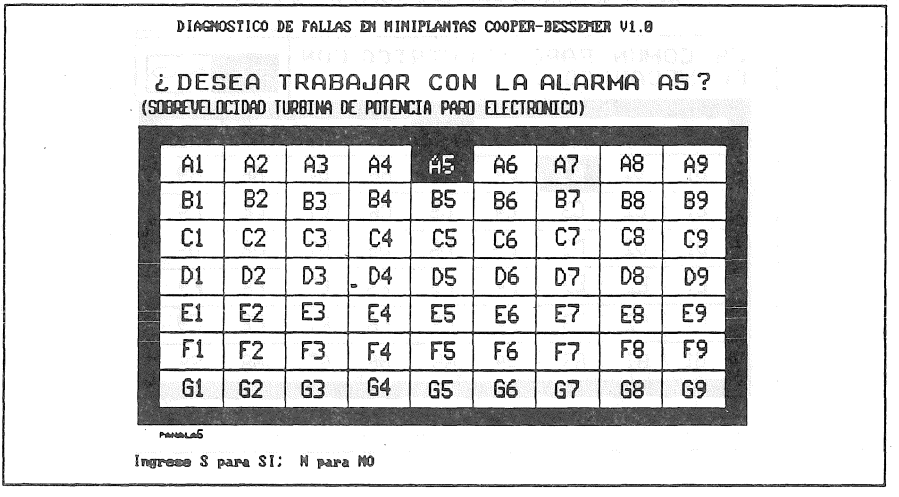


Figura 4
Pantalla de Confirmación.

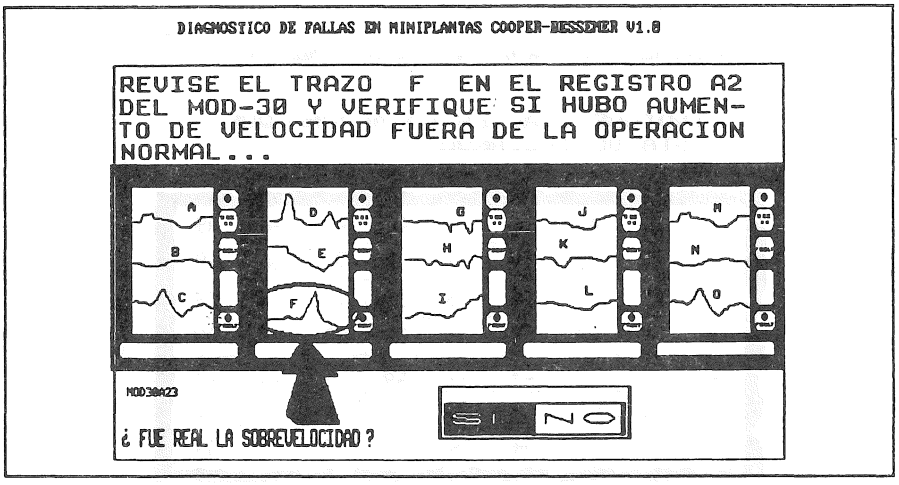


Figura 5
Revisión de Parámetro de Operación.

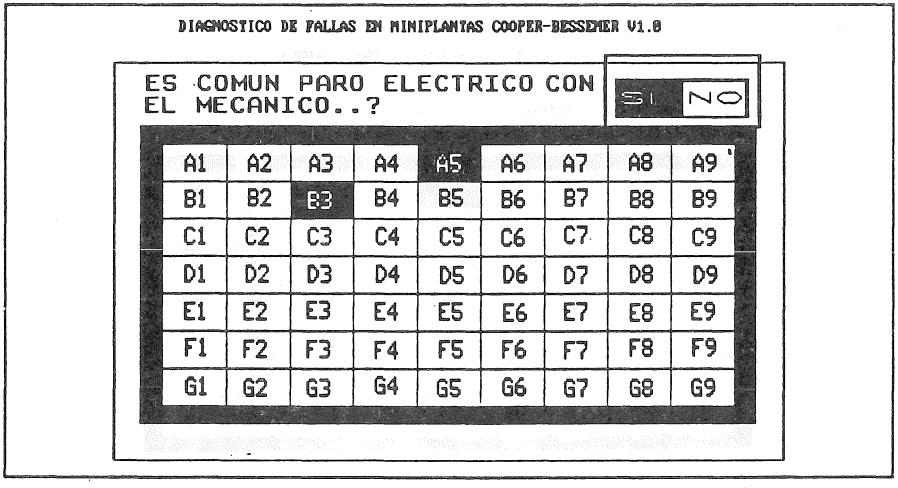


Figura 6
Verificación de Alarma Asociada.

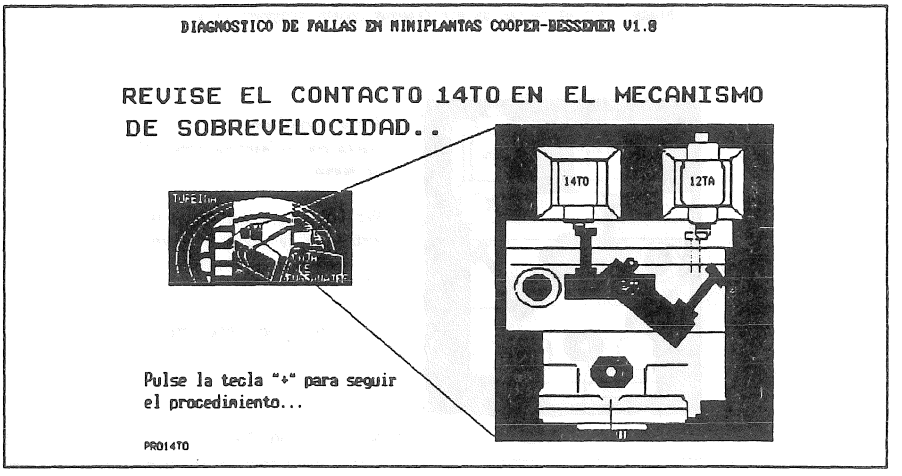


Figura 7
Diagnóstico de Dispositivo de Control.

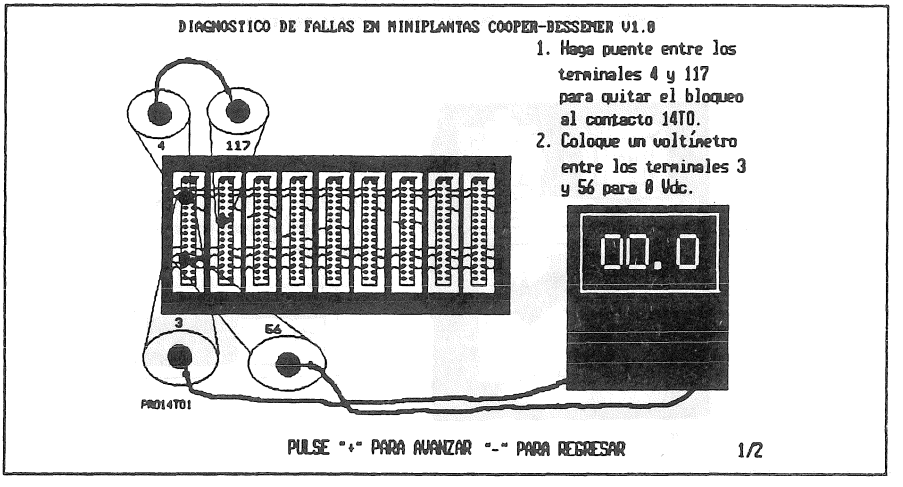


Figura 8
Procedimiento de Revisión (Paso 1).

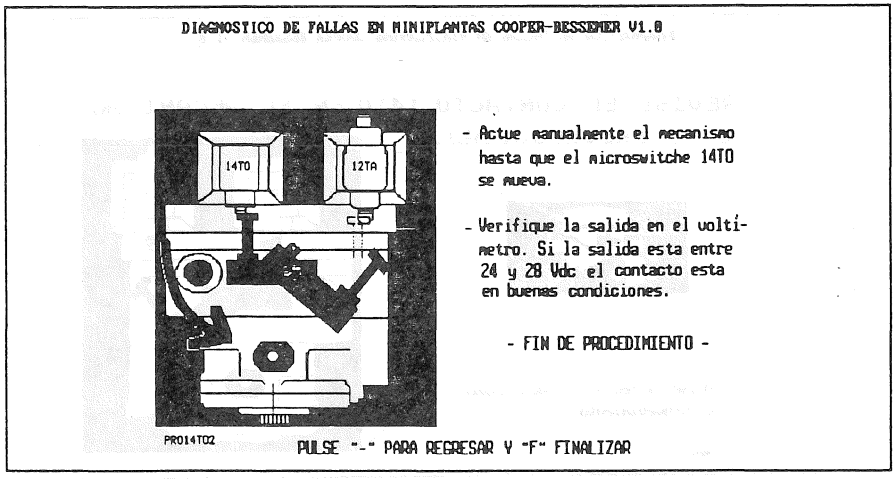


Figura 9
Procedimiento de Revisión (paso 2).

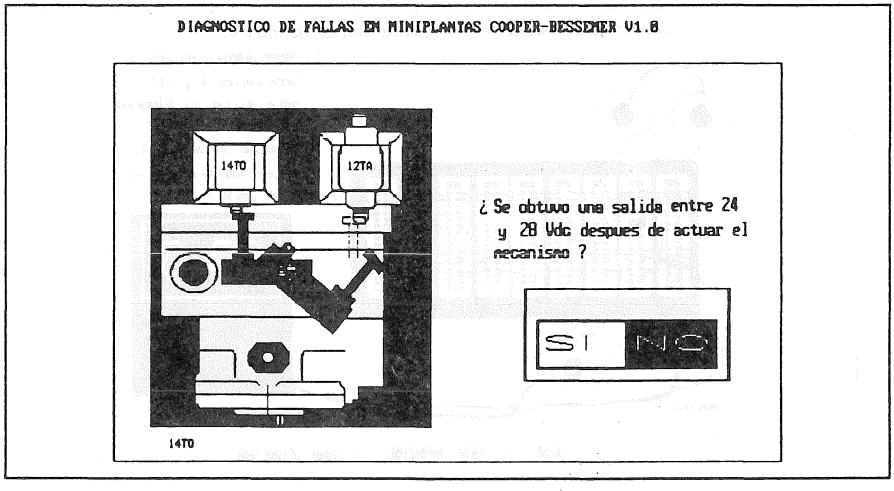


Figura 10
Confirmación de Diagnóstico.

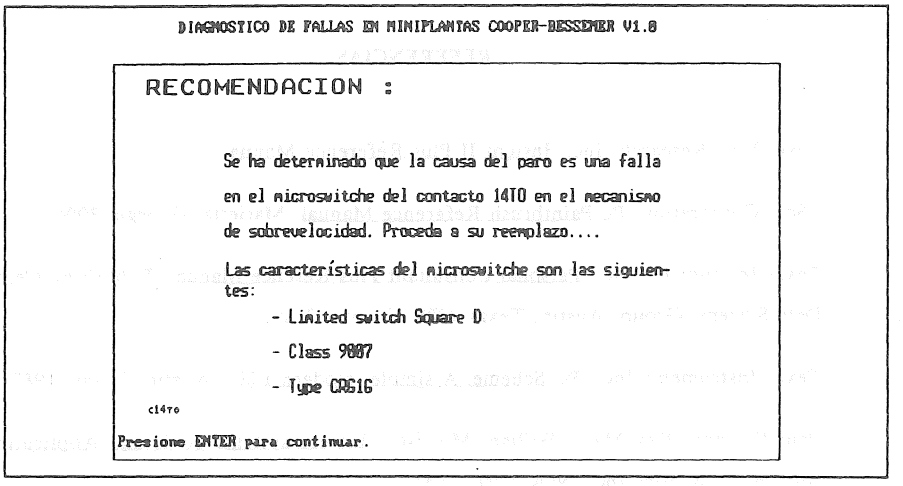


Figura 11
Fin de Consulta en *DIANA*.

REFERENCIAS

1.- Level Five Research, Inc., Insight II Plus Reference Manual.

2.- ZSoft Corporation, Pc Paintbrush Reference Manual, Marietta, Georgia 30067.

3.- Texas Instruments Inc., Personal Consultant Plus Refence Manual, Publishing Center, Data Systems Group, Austin, Texas 1988: Chapter 2, p.3.

4.- Texas Instruments Inc., PC Scheme: A simple, modern LISP, Austin, Texas, 1987.

5.- Paul Harmon, Rex Maus, William Morrisey, Expert Systems Tools and Applications, John Wiley & Sons Inc., New York, 1988.

6.- Karen L. Mc Graw, Karan Harbison-Briggs, Prentice Hall, Englewood cliffs, Knowledge Acquisition Principles and Guidelines, New Jersey, 1989.

7.- John Walters, Norman Nielsen, Crafting Knowledge Based Systems: Expert Systems made realistic, John Wiley & Sons Inc., New York 1988.

8.- Edward Feingenbaum, Pamela Mc Corduck, H. Penny Nii, The Rise of the Expert Company, Times Books, New York, 1988.

DISEÑO DE CIRCUITOS ELECTRONICOS BASADO EN REGLAS

Javier Cañas R. Ernesto Lux G.
Departamento de Informática
Universidad Técnica Federico Santa María

Resumen

Este trabajo describe un método de diseño de circuitos electrónicos basado en reglas, es decir, en especificaciones formales. Este método permite dos cosas importantes en el diseño de un circuito electrónico: simplificar su ciclo de vida facilitando las etapas de especificación, diseño y mantención. También permite su optimización al encontrar un espacio de soluciones que satisfacen los requerimientos.

La utilización práctica del método requiere sólo disponer de un lenguaje de programación basado en reglas tal como PROLOG u OPS5.

Se desarrolla un ejemplo práctico de diseño de un amplificador clase A con un transistor, en el cual dadas ciertas restricciones, se obtiene un espacio de soluciones completo.

1.- Introducción

Es tradicional en diversos campos de la ingeniería la siguiente afirmación: "el análisis es una técnica y el diseño un arte". Al profundizar en esto, nos damos cuenta que no es precisamente el arte lo que diferencia al análisis del diseño, sino más bien, una relación de complejidad.
Diseñar significa explorar múltiples, infinitas soluciones. Como en la práctica esto resulta imposible, es finalmente el criterio y la agudeza del ingeniero lo que determina la calidad de la solución.
Pero el diseño de circuitos electrónicos ¿tiene infinitas soluciones?. Si se asume que los componentes físicos que intervienen en un circuito electrónico están normalizados y sus valores son discretos, se logra un concepto distinto de diseño.
El diseño, desde esta perspectiva, se transforma en la obtención de un espacio de soluciones mediante un proceso combinatorio, el cual tiene que satisfacer las restricciones circuitales dadas para un diseño en particular. Utilizando técnicas de especificación formales de restricciones y ecuaciones circuitales, es posible obtener los espacios de soluciones buscados.

2.- El contexto del método

Un circuito electrónico, siempre se ubica en un contexto más amplio, es decir, un circuito es siempre un subsistema o un bloque de un sistema mayor.
Esto significa que estos sistemas están siempre jerarquizados y son factibles de descomponer en múltiples niveles.
Para ubicar el método en un contexto más general, se verá brevemente el ciclo de vida completo de un sistema electrónico.
El ciclo de vida de un sistema electrónico se compone de las siguientes etapas:

a) Análisis: Involucra entender el contexto del sistema.

b) Especificación: Involucra la especificación del sistema por sus características externas.

c) Diseño: En esta etapa se obtiene la topología y valores de componentes.

d) Construcción: Armado de un prototipo.

e) Prueba: Verificación del prototipo en laboratorio.

f) Validación: Revisión de las etapas anteriores verificando si al interconectar se cumplen las expectativas del análisis.

g) Mantención: Modificaciones posteriores al diseño.

El método propuesto permite facilitar las etapas de especificación, diseño y mantención.
Al formalizar las especificaciones, es posible lograr una mejor comprensión del problema, como también verificar que éstas sean consistentes.
Al tener especificaciones formales se automatiza el proceso de diseño.
La etapa de mantención también se simplifica ya que la especificación formal en sí constituye una documentación detallada del sistema electrónico.

3.- Restricciones

El método que se presenta contempla las siguientes restricciones:

i) Topología fija: El proceso de diseño consistirá sólo en obtener valores de componentes. Esto puede aparecer como una severa restricción, pero sólo significa trasladar el proceso de diseño a un ámbito particular ya que las topologías son también finitas. Para ejemplificar lo anterior considérese el caso de un amplificador con un transistor: existen diversas topologías pero cada una de ellas tiene características particulares de ganancia, impedancia y ancho de banda.

ii) Componentes de valores discretos: Los resistores, condensadores e inductores están discretizados según rangos de precisión y criterios de fabricación.
En el caso de fuentes de voltaje esto no parece ser así, pero en la práctica estos valores también son discretos, por ejemplo fuentes de 5, 10, 12, 15,... volts.

iii) Condiciones de borde formales: Las restricciones de diseño deben ser factibles de ser expresadas en forma matemática o lógica.

4.- El ambiente de diseño

El ambiente de diseño de un circuito electrónico lo forma:
a) Las leyes físicas que rígen su comportamiento (Leyes de Kirchoff).
b) Las relaciones terminales de los elementos.
c) El conjunto de componentes (resistencias, condensadores, etc.) estandarizados.
d) Los puntos a, b, c constituyen la base del conocimiento, y el punto d representa la especificación del circuito. Estos puntos determinan en definitiva el diseño del circuito electrónico, como se puede apreciar en la figura 1.

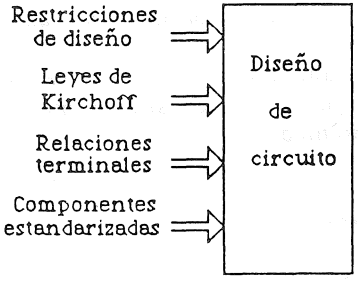


Figura 1

5.- Modelamiento lógico de un circuito

Tanto la base de conocimientos como las especificaciones de diseño se pueden especificar en forma lógica. Considérese a modo de ejemplo el circuito mostrado en la figura 2.

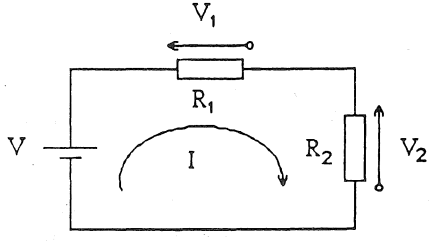


Figura 2

El conocimiento de este circuito está constituído por:

a) Leyes físicas:

$$V = V_1 + V_2$$

b) Las relaciones terminales:

$$V_1 = I * R_1$$
$$V_2 = I * R_2$$

c) El conjunto de componentes estandarizados:

$Q = \{ ri \mid ri$ es resistencia de valor normalizado$\}$
$R_1, R_2 \in Q$

El conocimiento se puede expresar mediante sentencias lógicas:

C_1 : $V = V_1 + V_2$
C_2 : $V_1 = I * R_1$
C_3 : $V_2 = I * R_2$
C_4 : $R_1 \in Q$
C_5 : $R_2 \in Q$

Suponiendo que $P = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5$:

entonces cualquier combinación de V, V_1, V_2, I, R_1 y R_2 que satisfaga P es una instancia válida y por lo tanto una solución del circuito.

6.- Diseño basado en reglas

Sea : $P = \bigwedge_{i}^{n} C_i$ donde C_i es la sentencia i de la base de conocimiento.

$R = \bigwedge_{i}^{m} R$ donde R es la restricción i.

El espacio de soluciones de un diseño se expresa por:

$$S = P \wedge R$$

O sea, es el conjunto de valores que satisfacen S

7.- Un experimento

Las ideas anteriores se ejemplifican en un diseño concreto: un amplificador clase, A de un transistor, polarizado con una fuente de 12 V. Las condiciones de borde están dadas por una resistencia de carga 0.5 KΩ y la fuente de señal tiene una resistencia interna de 0.6 KΩ. Se elige un transistor de silicio con ß mínimo de 100.
Se requiere de una ganancia de voltaje mínima de 32 y un swing de voltaje mínimo en la resitencia de carga de 2 volts peak.
La topología del circuito es la mostrada en la figura 3.

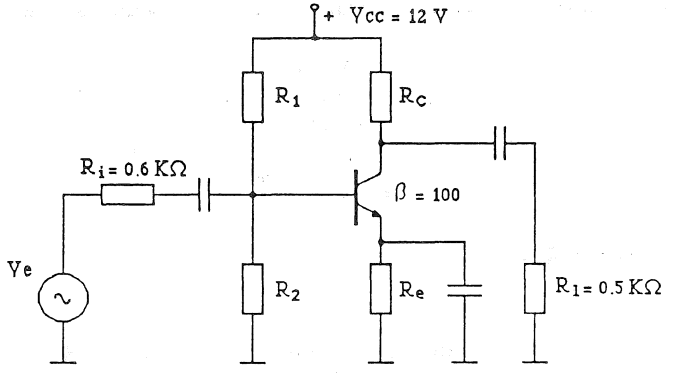


Figura 3

La base de conocimiento de este circuito lo constituyen las siguientes ecuaciones:

[1] $V_{bb} = V_{cc} * R_2 / (R_1 + R_2)$
(Debe cumplir con $V_{bb} > 0.7$ V)
[2] $R_b = R_1 * R_2 / (R_1 + R_2)$
[3] $I_{cq} = (V_{bb} - 0.7) / (R_e + R_b/ß)$
[4] $R_e > 10 * R_b / ß$
[5] $H_{ic} = ß * V_T / I_{cq}$
[6] $V_{ceq} = V_{cc} - I_{cq} * (Rc + Re)$
(Debe cumplir con $V_{ceq} > 0.2$ V)
[7] $swing = \min(V_{ceq}, I_{cq} * R_{ac})$
[8] $A_v = ß * (R_b \parallel H_{ie}) * (R_c \parallel R_L)/((R_i + (R_b \parallel H_{ic})) * H_{ic})$

Estas ecuaciones representan:

[1] Voltaje de base del transistor.
[2] Resistencia de base.
[3] Corriente de polarización de colector.
[4] Criterio de estabilidad de la red de polarización.
[5] Resistencia dinámica de entrada en el punto de polarización.
VT es una constante (25 mV @ 25 °C).
[6] Voltaje colector-emisor en el punto de operación.
[7] Swing de voltaje de peak en la carga. $R_{ac} = R_c \parallel R_L$.
[8] Ganancia de voltaje.

Las restricciones de diseño son:

[a] Av > 32
[b] swing > 2.0 V peak

De acuerdo al planteamiento original, el proceso de diseño significa explorar combinaciones válidas (resistencias normalizadas) de R_c, R_e, R_1 y R_2 que satisfagan las 8 ecuaciones y las dos restricciones anteriores.
De una manera más formal lo anterior se puede expresar en un programa PROLOG:

```
/*          DISEÑO AMPLIFICADOR CLASE A
Por simplicidad se omite en este programa el código
correspondiente al formato de salida de resultados. El predicado
principal se denomina "diseño"
*/

paralelo(Rp, Ra, Rb) :-                    /* calcula Rp = Ra || Rb  */

Rp is Ra * Rb / (Ra + Rb).

fuente_base(Vbb, Vcc, R1, R2) :-           /* ecuación  [1] */
Vbb is Vcc * R2 / (R1 + R2),
Vbb > 0.7.

estabilidad(Rb, Re, Beta) :-               /* ecuación  [4] */
Re > 10 * Rb / Beta.

i_colector(Icq, Vbb, Rb, Re, Beta) :-      /* ecuación  [3] */
```

```
Icq is (Vbb - 0.7) / (Re + Rb / Beta).

hie(Hie, Icq, Beta) :-                    /* ecuación [5] */
Hie is Beta * 2.5e-2 / Icq.

vceq(Vceq, Vcc, Icq, Rc, R1) :-           /* ecuación [6] */
Vccq is Vcc - Icq * (Rc + Re),
Vceq > 0.2.

swing(V1, Vccq, Icq, Rc, R1) :-           /* ecuación [7] */
paralelo(Rac, Rc, R1),
Vceq > Icq * Rac,
V1 is Icq * Rac | V1 is Vceq.

ganancia(Av, Rc, Rb, Ri, RL, Hie, Beta) :-  /* ecuación [8] */
paralelo(Rin, Rb, Hie),
paralelo(Rout, Rc, RL),
Av is Beta * Rout * Rin/((Ri + Rin) * Hie).


/*          BASE DE DATOS RESISTENCIAS
Esta base de datos corresponde a resistencias con 10% de
tolerancia. Por simplicidad se muestra sólo una parte de esta base
de datos.
Es conveniente mantener estos datos en un archivo aparte.
Los valores están expresados en KΩ. */

resistor(10.0e-2).                        resistor(15.0).
resistor(12.0e-2).                        resistor(18.0).
resistor(15.0e-2).                        resistor(22.0).
resistor(18.0e-2).                        resistor(2.7e1).

                .........................
/* esta es la regla principal */
diseño(Av, Swing, Vcc, Beta, Rc, Re, R1, R2, Ri, RL) :-
resistor(R1),
resistor(R2),
paralelo(Rb, R1, R2),
fuente_base(Vbb, Vcc, R1, R2),
resistor(Rc),
estabilidad(Rb, Re, Beta),
i_colector(Icq, Vbb, Rb, Rc, Beta),
resistor(Rc),
vceq(Vceq, Vcc, Icq, Rc, Re),
swing(V1, Vceq, Icq, Rc, Re),
V1 >= Swing,
```

hie(Hie, I_{cq}, Beta),
ganancia(G, R_c, R_b, R_i, R_L, Hie, Beta),
G >= A_v.

El espacio de soluciones obtenido después de ejecutar este programa es:

Diseño para Ri=0.6K, Rl=0.5K, Av > 32, swing > 2.0, Vcc=12V, ß=100

Av	Swing	Rc	Re	R1	R2
32.03	2.23	0.68	0.18	6.80	2.20
32.34	2.18	0.82	0.22	6.80	2.20
32.15	2.10	1.00	0.27	6.80	2.20
32.32	2.68	0.82	0.18	8.20	2.20
32.23	2.55	1.00	0.22	8.20	2.20
32.96	2.17	0.82	0.22	8.20	2.70
32.88	2.06	1.00	0.27	8.20	2.70
32.11	2.07	1.20	0.33	8.20	2.70
32.75	2.64	1.00	0.22	10.00	2.70
33.27	2.19	1.00	0.27	10.00	3.30
32.57	2.18	1.20	0.33	10.00	3.30
33.09	2.09	1.00	0.33	10.00	3.90
32.95	2.03	0.82	0.33	10.00	4.70
32.60	2.62	1.20	0.27	12.00	3.30
32.61	2.45	1.20	0.33	12.00	3.90
32.00	2.12	1.50	0.47	15.00	5.60
32.71	2.17	1.20	0.47	15.00	6.80

8.- Comentarios

Si bien el ejemplo presentado es un circuito relativamente simple, se ha seleccionado ya que contiene varios compromisos de diseño que son difíciles de satisfacer. Por ejemplo, la ganancia (ecuación [8]) depende del valor de H_{ic} (ecuación [5]) que a su vez depende de I_{cq} (ecuación [3]). También el swing (ecuación [7]) depende de los valores de I_{cq} y de Rc.

Normalmente al diseñar en forma manual se recurre a algunas simplificaciones, como por ejemplo, el darse en forma arbitraria algunos valores de resistencias. Otra simplificación usada es descomponer la expresión de ganancia en factores iguales [3].

En el diseño manual los componentes se normalizan una vez que se ha calculado su valor. Esto significa sucesivas iteraciones del proceso de diseño para intentar cumplir las especificaciones.

El programa señalado es completamente general siendo su única restricción la topología del circuito. El programa PROLOG permite experimentar diversas combinaciones de parámetros tales como: fuentes de polarización, valores de ß, resistencias, etc.
Fue desarrollado en Quintus Prolog en una máquina Vax Station 2000 bajo el sistema operativo Ultrics.
El tiempo de ejecución para obtener el espacio de soluciones completo, es decir, las cuatro resistencias de incógnita es de 110 minutos aproximadamente, en un programa compilado.

9.- Conclusiones

El diseño de sistemas de ingeniería sometido a múltiples compromisos y restricciones, se puede encarar desde este nuevo enfoque, el cual al estar basado en reglas, traslada la concepción analítica tradicional en lógica que convierte lo algorítmico en declarativo.

El método resulta especialmente atractivo en el diseño de circuitos electrónicos. Un uso importante es en la docencia de ingeniería electrónica debido que al expresar un circuito en reglas, se obtiene una mejor comprensión de él y también una excelente documentación. El programa de diseño son sólo las ecuaciones que describen el circuito expresadas en PROLOG. Otro uso atractivo es la optimización. Al generarse un espacio de soluciones es posible optimizar en forma visual o añadir mayores restricciones.

Un punto débil es la complejidad temporal del programa, lo que establece automáticamente una cota en cuanto a la complejidad circuital. Frente a este hecho hay dos reflexiones: se puede introducir un predicado de optimización que mediante alguna heurística reduzca el espacio de búsqueda, y lo otro es que también siempre es posible descomponer un circuito en sub-circuitos y realizar optimizaciones locales.

Referencias

1) P. Horowitz, W. Hill: "The Art of Electronics", 2 edición, Cambridge University Press, 1989.

2) G. Robison: "An Introduction to Mathematical Logic", Prentice-Hall, Englewood Cliffs, N.J., 1969.

3) D. Schilling, C. Belove: "Electronic Circuits: Discreted and Integrated", McGraw-Hill, New York 1968.

4) Quintus Computer Systems, Inc.: "Quintus Prolog Reference Manual", 1986.

5) Clocksin & Mellish: "Programming in PROLOG", Springer Verlag, NY, EEUU, 1984.

6) Hayes, F. "Rule-base system", Communications of the ACM, 28:9 (Sept. 1985).

Knowledge Acquisition Combining Analytical and Empirical Techniques

Ramón Sangüesa [2] **Mario Martín** [1] **Ulises Cortés**[1]

(1) Facultat d'Informàtica de Barcelona (UPC)
Departament de Llenguatges i Sistemes Informàtics (it)
Pau Gargallo, 5. Barcelona. 08028. Spain
e-mail: fibdls39@fib.upc.es ia@fib.upc.es

(2) Escola Universitaria d'Informatica de Lleida
Universitat de Barcelona
Estudi General de Lleida. Rambla d'Arago 307. 25006 Lleida. Spain
e-mail: fibdls02@fib.upc.es

ABSTRACT: In this paper we introduce a methodology for classification-oriented knowledge-base (KB) generation using LINNEO, a software for fuzzy classification and rule generation which resorts to analytical (Explanation Based Generalisation -EBG-) and empirical (Similarity Based Learning -SBL-) knowledge acquisition techniques. LINNEO builds a classification from a set of (frequently noisy) observations and a (possibly incomplete) domain theory supplied by the expert. The final result is a MILORD knowledge base. MILORD is a knowledge base generator which uses fuzzy rules for the management of uncertainty. It is believed that integrating both types (EBG, SBL) of learning techniques improves the whole process of knowledge acquistion. In our approach SBL is biased by a domain theory which helps in focusing the induction process. This is seen as a major advantage, overcoming the usual drawbacks of SBL techniques.

KEYWORDS: Automatic knowledge acquisition, machine learning, classification, expert systems, knowledge engineering, SBL, EBG, fuzzy logic.

1. INTRODUCTION.

One of the major problems in developing expert systems is knowledge acquisition. A variety of techniques have been devised in order to solve the "knowledge engineering bottleneck". A possible approach consists in using automated knowledge acquisition tools. These, in turn, make use of machine learning techniques, mainly Similarity Based Learning (SBL) and Explanation Based Generalisation

(EBG) [DEJO86], which stress two different paradigms in learning: inductive methods and deductive methods.

Inductive learning methods extract new hypothesis from observations but they need a great number of examples to be effective, they are generally blind to the relevance of certain attributes and the knowledge they finally infer may be erroneous. On the other hand, deductive methods ensure that new knowledge is logically sound. Moreover, they need much less examples. However, a consistent and complete domain theory is needed right from the beginning of the process which is not usually the case.

When one compares both approaches, one feels that, in a certain sense, they are complementary. It seems sensible to combine both knowledge acquisition techniques in order to overcome their disadvantages. Here we discuss LINNEO, a knowledge acquisition methodology in classification domains which integrates both techniques. On one hand, it uses - possibly noisy- observations to induce classes. On the other, it takes advantage of a domain theory, which represents the expert's current state of knowledge (so it may be incomplete), this theory constraints the possible outcomes of the classification process.

Figure 1 shows the general framework of the methodology. The human expert abstracts a collection of observations (sample) from the problem domain that he thinks of as relevant. He also selects a set of attributes (descriptors) that he assumes to be significant for the characterization of the domain objects. A sample is then modelled by a set of attribute/value vectors. At the same time, the expert can express what he already knows about the domain. This knowledge (that we will call Domain Theory) is represented as constraints and/or examples on classes already known to exist. Starting with this knowledge and informations LINNEO produces a classification. It uses induction over the observations taking into account the domain theory available which constrains the inductive process. The results of this classification are an intensive and extensive description of the generated classes and a fuzzy membership matrix that relates observations to generated classes. The expert evaluates the results and eventually modifies some attributes or some aspects of his domain theory. When a correct classification is obtained, rules are derived from the generated classes. Each rule includes in its antecedent the membership conditions and a fuzzy certainty factor for a particular class. Each antecedent is a conjunction of logical conditions on the values that descriptors must show to belong to the class whose identifier appears on the rule's consequent. These rules are analyzed and, through subsumption detection and synonymy analysis, they are organized in a hierachical way.

The results are made known to the user, who can accept and confirm them. He also has the chance to go back to the classification module, the definition of the Domain Theory or to the sample selection step and obtain another results. When all these actions are over, rules can be validated using new observations not previously in the sample. As soon as the expert accepts the resulting rule

set, it is transformed into MILORD [SIER89] rules and modules.

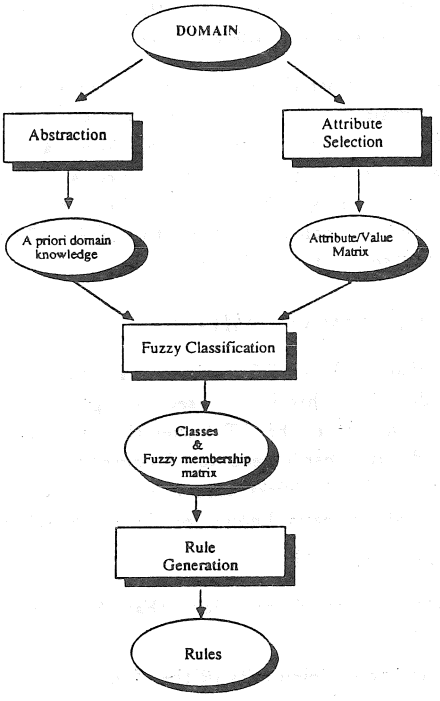


Figure 1. LINNEO: Overview of the different steps involved

2. ATTRIBUTE AND OBSERVATION SELECTION.

This is a very important step because it determines the quality of the results. Induction depends crucially on the quality of the given set of observations. In this step, the expert defines a set of observations that he/she thinks of as sufficient to model the domain and also defines a complete set of attributes relevant to the classification goal intended.

The attribute selection is essential because it determines the set of properties that define the set of objects. A poorly informative or barely discriminant attribute set returns an incorrect result. It should be noted that these properties will become the vocabulary used by the final expert system.

The expert is allowed to represent attributes by means of two pre-defined types:

- Quantitative: measurable properties. For example, in macroecnonomic classification, the national income, etc.
- Qualitative: properties that cannot be ordered. For example, the colour of an object, the marital status of someone, etc,.

It is also necessary to choose those observations considered important for the modelization of the domain. It is intended to cover all the possible prototipical cases. As it happened in attribute selection, an inadequate or incomplete set of observations may lead to a poor, when not erroneous, result. For each observation, a vector is defined whose length is just the number of attributes. This vector is the value vector for each property of the object. Then, we have a set of vectors representing the set of observations.

3. EXPRESSING THE DOMAIN THEORY.

In this step a domain theory (DT) expressing what the expert can explain about the domain is defined. This DT is used as a group of constraints guiding the inductive process. Therefore, the DT biases the set of possibles classes. This DT, just acts as a guide; it doesn't need to be complete.

The expert is allowed to express this theory in terms of constraints and examples on classes he/she already knows for sure that exist:

- Class name: an identifier.
- Constraints on classes: set of conditions that elements must fulfill in order to belong to the class.
- Examples: a list of typical elements of the class.

Conditions appearing in constraints need not correspond to the attributes used in defining observations. However, there should exist a mapping between the former and the latter. This mapping is expressed by means of a dictionary which can roughly be understood as a set of rewriting rules. On the other hand, examples have to be a subset of the previously defined observations.

The importance of definining these constraints and examples is related to its usefulness in guiding the classification process.

4. CLASSIFICATION STEP.

Once the expert has selected the descriptors and the observation sample and has defined the corresponding constraints (in case he/she knows them), classification starts. This process induces a tentative conceptual structure for the domain. In general, any inductive classification process will group objects into classes using some criterion on similarity (SBL technique). In our approach, constraints on *a priori* classes are added to this similarity criterion.

We also want to set similarity within the framework of fuzzy sets theory. With these ideas in mind we have decided to use the classical concept of distance as a fuzzy similarity value. If we take d* as the normalized distance (d) then s*, is defined as:

$$s^* = 1 - d^*$$

that is, the fuzzy similarity value. Note that fuzzy similarity and distance are mutually inverse: the greater the distance the lower the similarity.

As we have seen in section 3., objects are represented as numeric/symbolic vectors of length p, p being the number of descriptors in the observation matrix. Position i of vector j shows the symbolical or numerical value of descriptor i of object j. The distance that will be used in determining the similarity between two objects is the general Hamming distance so that the distance between object O_i and object O_j is:

$$d(O_i, O_j) = \sum_{k=1}^{n} (dif(O_{ik}, O_{jk}))$$

Where:

- dif (O_{ik}, O_{jk}): if k is a qualitative property, its value is 1 if O_{ik} and O_{jk} are different modalities and 0 otherwise.
- dif (O_{ik}, O_{jk}): if k is quantitative is the absolute value of their difference

Two aspects are taken into account when representing a class: its extensional description and its intensional description. The first one is given by the enumeration of all elements contained in the class. The second one is a vector with p properties containing also the class centre and the standard deviation. Let's see an example:

```
(WEALTHY-COUNTRIES)
EXAMPLES: (W-GERMANY)
CONSTRAINTS:
    (AND (> EXECUTIVE-SAFETY 0.6) (INVESTMENT-TREND EXCELLENT))
ELEMENTS:
    (SOUTH-AFRICA W-GERMANY AUSTRALIA CANADA DANMARK JAPAN
NORWAY NETHERLAND SINGAPUR SWITZERLAND)
CENTRE:
    ((EXECUTIVE-SAFETY 0.8019)
    (FINANCIAL-SAFETY 0.7784)
    (POLITICAL-SAFETY 0.7858)
```

(PRESENT-INVESTMENT (LOW 0.2) (GOOD 0.2) (STABLE 0.6))
(INVESTMENT-TREND (LOW 0.0) (GOOD 0.0) (STABLE 0.0) (EXCELLENT 1.0))))
STANDARD DEVIATION: (0.2 0.1 0.12 NIL NIL)
MAXIMUM-MINIMUM: ((0.953 0.679) (0.896 0.712) (0.822 0.757) NIL NIL)

This is a example of an *a priori* class. Its existence, name, two constraints and one example are known and have been explicited by the expert. The rest of the fields are completed during the classification process.

The center of the class is obtained by calculating the mean value for each quantitative property of every object. For qualitative properties, the center includes each one of their modalities with its corresponding occurrence frequency. Note hat the center of a class can be considered as a prototype of the objects contained in the class. Therefore, it is possible to compare any object against a class. The distance between an object and a class can be taken as the inverse of the degree of fuzzy membership of the object to the class.

If a value for the distance is set as the limit for an object to belong to a class (limit that we will call radius), then the intended classification rule (expressing the similarity criterium with the DT) is found:

IF *the distance between a given object and the center of a class is less than the class radius* **AND** *it fulfills the class constraints*
THEN *it can belong to the class*
ELSE *it cannot belong to it.*

In this context, the radius r can be interpreted as the degree of selectivity of the classification. If the radius is large, there will be few classes with many observations in each one. If it is small, there will be many classes containing few observations. A tradeoff has to be found between large clusters of observations (large radius) and a large number of classes avoiding low-similarity elements to be clustered together (small radius). Frequently, the expert will have to create several classifications with different radiuses until he/she being able to choose one of them.

Taking this criterion into account we have developed the algorithm that we use for classification. Its outline is as follows:

Initialization:

- *Load the classes a priori defined by the expert.*
- *Verify that the examples of the classes fullfill the decision rule.*
- *Initialize the centre of the a priori classes with the examples. The initial center of a class is just a vector with the mean of the properties of the examples.*

Body: For each one of the objects in the system the best class among the current ones is selected:

- *Select those classes whose constraints are not violated by the object.*
- *The best class for an object is taken to be the one in the previous set of classes with minimum distance to the object. Two things can happen at this moment:*

  a) Distance to the best class is less than the current classification radius. In such case, the object is included in the class and the center of the class is recalculated. While recalculating the centre some objects may go away from the centre and locate themselves farther than the radius. If this happens, these objects are eliminated from the class to be be reclassified later but this time without modifying the centers.

  b) Distance to the best class is greater than the current classification radius or there exists not a best class. In such case a new class is created. The vector of the object currently under consideration becomes the initial centre of the class.

As it can be seen, induction is guided by the DT expressed as constraints. In our apporach, inductive (SBL) and deductive (EBG) are the extreme points of a single continous axis. At one extreme, when no DT is available, the algorithm works as a pure SBL method. At the oppposite one, when a complete and consistent DT is at hand, the algorithm is merely a deductive technique (EBG) with no induction at all. At the middle of this axis we have the situation when induction is biased by a DT (possibly not complete).

When the expert agrees with no classification, he/she will probably have to modify the set of observations, the set of atributtes or perhaps the domain theory previously defined. This is a usual situation that corresponds to a cyclical interaction between the user and the system in order to define effectively the domain. Verification of classification quality can be done with a collection of functions that calculate distances between class centers (in order to spot class overlappings), between centers and objects (to see if some element located under the radius of two classes is badly assigned), etc.

Similarly, besides expert opinion, several criteria can be used to decide when a given classification can be tagged as "good":

- Every class contains a similar number of observations.
- The number of useful descriptors is small.
- Significative distance exists between class centers (greater than their radiuses)
- Few overlappings (observations falling under two classes)
- etc.

The result, once it has been confirmed by the expert, is a list of classes and a fuzzy membership matrix. This two items of information are then transferred

to the next module wich will turn them into classification rules.

Once a partition over the observations is obtained, the same algorithm can be applied on the class centers, obtaining then a superclassification, a hierarchical classification tree.

The classification module here discussed has several advantages:

- It takes advantages of empirical and analitical techniques
- It obtains in a fast way a first aproximation to the domain structure.
- It works incrementally: addition of new data does not imply reclassification of previous observations.
- It deals simultaneously with numerical and categorical attributes.
- It works with prototypes (and an intensional description of classes).

5. THE RULE GENERATION MODULE

This module obtains a set of classification rules which have to mimic as approximately as possible the classifier's behaviour. The generated rules have the following structure:

$$(R_i, A, C, \alpha)$$

where,

R_i: rule identifier

A (antecedent): A logical conjunction of conditions that have to be met by observations to be included in a given class. For example,

$$(color(red)) \wedge (weight(high))$$

C: (consequent): The identifier of one of the classes previously obtained.

α : the degree in which the observations belong to the class. It is the certainty value associated with the rule's conclusion.

In order to obtain the rules, the description of classes and the fuzzy membership matrix created by the classfier are used. These two information items are the final result of the classification module. The first one allows us to elicitate which conditions have to be met by any observation to be included in each class. The second one allows calculation of the certainty level at which one can say that the observation belongs to a class. Recall that the center of a class is a vector C whose dimension is equal to the number p of properties defined by the expert. Each property has an associatetd value. In the case of a quantitative property,

the corresponding value of the centre is the mean value of this property calculated over all objects in the class. For qualitative properties the center contains each one of it modalities and the corresponding frequency within the class. This frequenciy gives an idea about how significant is the modality within the class. The "center" indicates which values must have the properties for a given object to be considered as part of the class.

The rule generator turns the values of the centre into logical expressions that correspond to the antecedent of a rule. In the case of qualitative properties this transformation is straightforward A predicate is created. Its name is the same as the property. This predicate is applied to the value of the modality appearing in the centre. If, for example, we have a series of observations about esponges and one of the characteristics used is "cortex" with modalities "present" and "absent", they will be transformed into the predicates `cortex(present)` and `cortex(absent)`. Of course, this conversion will not be always made. Previously one has to decide if this characteristic is relevant or not. The heuristic we use in detecting the relevance of descriptors is very simple but effective. In principle, the center always being in normalized form, we consider values above some value as "high" and under some other value as "low". Both values have been previously set by the expert. Then, values above the first value and under the second one are taken to be significant.

It is important to realize that values very near to 0 may suggest that what is really significant is the absence of the characteristic they refer to. Therefore, they will be transformed into negated predicates. For example, in the country classification example, if we consider the "trend" property whose modality "excellent" has a normalized value of 0.0 the corresponding transformation is: `(trend(not(excellent))`. Finally it has to be noted that negative characterization of values near zero is not always necessary. This is done when no other significant descriptor has been found or when anbiguities arise between two or more rules sharing the same significant descriptors and the negative characterization is the only way to discriminate between them.

Uncertainty factors assessment:

Each rule has an associated degree of certainty, calculated as the degree of membership to the class of observations that it is characterized by the rule's antecedent. This information is drawn from the fuzzy membership matrix previously obtained by the classification process. If the number of objects is n and the number of classes m, the fuzzy membership matrix is an $n \times m$ matrix of numerical values ranging form 0 to 1. Theses values show the degreee of membership for

every object to every class.

	Class 1	Class 2	Class 3
Elem. 1	1.0	0.2	0.1
Elem. 2	0.6	0.3	1.0
Elem. 3	0.4	1.0	0.8
Elem. 4	1.0	0.8	0.45
Elem. 5	1.0	0.8	0.7

$\alpha = 0.55$

Figure 3. Fuzzy membership matrix.

For an α value of 0.55 the elements of the different classes are shown by a shadowed square. Of course, the interesting α values are those that induce a change in the structure of classes. In general, the relation between class aggregation and the α values is given by a decreasing nonmonotonic function. Starting with an α value that ensures a convenient certainty (0.55, for example) and incresasing it is easy to see that the structure of classes changes, reducing the number of their elements. Using this property we get rules of this kind:

$$(r_1, A_1, C_1, 0.55)$$

$$(r_2, A_2, C_1, 0.65)$$

$$(r_3, A_3, C_1, 0.85)$$

$$(r_4, A_4, C_1, 0.95)$$

$$(r_5, A_5, C_1, 1.0)$$

being the α values precisely those values that changed the structure of the class. All these rules point to the same class. Their antecedents, however, are differnet because their components have also changed with α and, accordingly, the calculated centre for each α has been different. This has originated different antecedents, that is, different membership conditions for each class. In general the lower the α value the less strict the conditions will be. In other words, rule specificity increases with the α value. Having several rules with increasing certainty for the same class gives us a very interesting information. Less certainty

implies greater generality. This is the base for organizing rules in a hierarchical way. The fundaments of the process are described in [MART90a].

Analysis and diagnostic

As we explained in the introduction, the rule generation module offers several tools for analysis and diagnostic. The expert can use them to refine the rules in the rule generation module or he/she may choose to go back to the classification module. The analysis currently made by the rule classification module are:

- Synonymy between groups of descriptors. The distances bewtween descriptors are found. If there is zero distance between a pair of descriptors, they are considered to be synonymous. The user is asked to confirm this and a name is requested for the new substituing descriptor.
- Rewriting of equivalent descriptors. Sometimes, it is possible to check if the distances between several descriptors and another different one is very small. After user's confirmation the second one substitutes the first group.

To sum up, the rule generation process involves the following steps:

Certainty factors calculation

Starting from the class representation generated by the classifying module and from the fuzzy membership matrix:

$$(clase_1(c_1, ..., c_m, q_1, ..., q_n))$$

$$(clase_k(c_1, ..., c_m, q_1, ..., q_n))$$

$c_1, ..., c_m$ being the values of quantitative properties $q_1, ..., q_n$ being those of qualitative properties.

Following iterating over the α values, several centres are obtained for each class and an "intermediate" rule representation is generated.

$$((c_1, ..., c_m, q_1, ..., q_n)\alpha_{11}\, class_1)$$

$$((c_1, ..., c_m, q_1, ..., q_n)\alpha_{12}\, class_1)$$

$$((c_1, ..., c_m, q_1, ..., q_n)\alpha_{j1}\, class_j)$$

$$((c_1, ..., c_m, q_1, ..., q_n)\alpha_{kq} class_k)$$

Significant property selection

Rules in intermediate representation are scanned marking all descriptors whose value is above or under the levels set by the user. A new set of rules is obtained, each rule now having a marked descriptors list added.

$$((c_1, ..., c_m, q_1, ..., q_n)\alpha_{kq} class_k(c_3\, c_5\, q_2\, q_5))$$

Antecedent ambiguity elimination

All pairs of rules that share the same marked descriptors are found. A search for discriminating descriptors is started. First, descriptors with value near to zero appearing in one rule but not in the other are looked for. If they are found, they are inserted in the marked descriptors list. If the ambiguity cannot be solved, the user is warned and may review the current classification starting againg classification and rule generation with new parameters.

Antecedent simplification

Once ambiguities are solved, antecedents are substituted by the marked descriptors lists. Similarly an interactive process starts trying to cluster mutually synonymous descriptors (through distance analysis). If they are found, and after user's confirmation, a rewriting rule of the following form is created:

$$(c_1, c_2, ..., q_j) \Longrightarrow (c_i, c_k, q_r)$$

where $\Longrightarrow$ is the rewriting symbol so that any new rule having $(c_1, c_2, ..., q_j)$ in its antecedent will be changed having as new antecedent (c_i, c_k, q_r).

Transformation into predicate representation

After obtaining an adequate set of nonambiguous rules, the transformation into predicate representation can start. The final result is a collection of rules of the form:

$$(pred_1(value_1) \wedge pred_2(value_2) \wedge ... \wedge pred_3(value_3), \alpha, class_i)$$

These rules are then translated into MILORD formalism becomig a knowledge base that can be directly used by this expert system shell.

6. CONCLUSIONS

LINNEO, a software for rule generation in classification domains has been discussed. Its main contribution is the integration of two different learning techniques in knowledge acquisition. Integration tries to overcome the shortcomings of SBL and EBG techniques when acting on their own. With LINNEO the expert doesn't need to have a complete and fully consistent theory of the domain to start gathering knowledge. Nevertheless, the expert is able to express any previous knowledge he/she may have about the structure of the domain. This can be done by means of constraints and examples. In the former state the constraints are necessary conditions for objects to belong to a given (and known to exist) class. The latter facilitate the characterization of prototypical description of classes. In our view SBL and EBG can be seen as complementary poles in the same single axe. LINNEO II allows to use pure extremes or mixed techniques according to the explicited knowledge by the expert.

LINNEO has been successfully applied to sponge classification [DOMI90], mental illness classification [ROJO91] and economical classification [MART90b].

7. REFERENCES

[AGUI86] Aguilar, J., Piera, N. *"Les connectifs mixtes: de nouveaux operateurs d'associations des variables dans la classification automatique avec apprentisage"* In: **Data Analysis and Informatics**, edited by E. Diday. Elsevier Science Pub. (1986). pp. 253-265.

[CORT86] Cortés, U., López de Mántaras, R., Agustí, J., Plaza, E., *"Fuzzy knowledge engineering techniques in scientific document classification."* ACM Sigart. International Symposium on Methodologies for Intelligent Systems pp.94-112. Knoxville, Tennesse.

[DEJO86] DeJong, G.F, Mooney, R.L. *"Explanation Based Learning: an alternative view"* Machine Learning, 1 (1986).

[DOMI90] Domingo, M. *"Aplicació de tècniques de I.A. (LINNEO) a la classificació sistemática: O. Hadromerida (Demospongiæ.Porifera)"* Master Thesis. Ecology Dept. University of Barcelona (1990).

[LOPE89] López de Mántaras, R., Plaza, E. *"Model-based knowledge acquisition for Heuristic Classification Systems"* SIGART Newsletter. Special issue on Knowledge Acquisition, 108 (Abril 1989).

[MART90a] Martín, M. *CCA: Conceptual Connectivity Analysis. Une application pour l'analyse de descriptions conceptuelles floues.* Report Laboratoire d'Automatique et Analyse de Systemes. Toulouse, 1990 (in press).

[MART90b] Martín, M. Sangësa, R. *LINNEO: Herramineta para la adqusición de conocimientos y generación de reglas en dominios de clasificación y diagnosis.* IBERAMIA-90. Morelia, MEXIC, 1990.

[MICH85] Michalski, R, Steep, R. E. *"Learning from observation: Conceptual Clustering"* Machine Learning an A.I. perspective, Tioga, Palo Alto, CA. (1983).

[PIER87] Piera, N. **Connectius de lògiques no estandàrd com a operadors d'agregació en classificació multivariable i reconeixement de formes.** Ph.D. Facultat d'Informàtica de Barcelona. Universitat Politècnica de Catalunya. Barcelona (1987).

[PLAZ87] Plaza, E. **Sistema d'ajut a l'adquisició i estructuració de coneixements** Ph.D. Facultat d'Informàtica de Barcelona. Universitat Politècnica de Catalunya. Barcelona (1987).

[QUIN83] Quinlan, J.R. *"Learning Efficient Classification Procedures"* **Machine Learning an A.I. perspective**, Tioga, Palo Alto, CA. (1983).

[ROJO91] Rojo, E. *"Aplicació del software LINNEO a la classificació d'enfermetats mentals"* Ph.D. (forthcoming) University of Barcelona (1990).

[SIER89] Sierra, C. **MILORD. Arquitectura multi-nivell per a sistemes experts en classificació.** Ph.D. Facultat d'Informatica de Barcelona. Universitat Politècnica de Catalunya. Barcelona (1989).

[ZADE65] Zadeh, L. *"Fuzzy sets"*. Information and Control, 8, pp. 338-353, (1965).

Programming Environment for Expert System Shell

Wided LEJOUAD

SECOIA Project

INRIA/CERMICS Sophia-Antipolis

2004, Route des Lucioles

F-06561 Valbonne Cedex (France)

lejouad@mirsa.inria.fr

March 8, 1991

Abstract

The frequent use of object-oriented systems drove us to think about an interactive programming environment generator, able to deal with these systems thanks to specification languages. The first step of this project was to take an example of programming environment generator, such as **Centaur**, and to see its behaviour with the **Smeci** multiformalism expert system shell. Many problems have been studied for modeling class hierarchies and inheritance, type-checking rules with respect to class definitions. From this work, two editors are issued: one syntactic editor for the rule language and another for the class language, and a program that type-checks the rules in relation to class definitions.

Keywords: object-oriented systems, expert system shell, inheritance and programming environment.

1 Introduction

Current application diversity and complexity have driven computer research to turn towards a new programming style : object-oriented programming [MNC+89]. It makes flexible and easy the resolution of many problems in different computer fields.

Moreover, the frequent use of object-oriented systems has led researchers to think about an interactive programming environment generator, able to treat these systems thanks to specification languages, as it's the case with classic programming languages.

The considered interactive programming environment generator and object-oriented system are respectively Centaur and Smeci. So, we want to study the feasibility of the second one environment under the first one [Lej90].

2 Tool description

In this section, we describe the tools used to achieve this work. Our objective is to familiarize readers with some notion of expert system shell and interactive programming environment generator.

2.1 Smeci

Smeci [II89] is a shell which allows expert system development and adjustment. It is suited to the realization of expert systems in design, planning and complex diagnosis. It is based on three principles:

- The representation of factual knowledge in object form.
- The representation of deductive knowledge in rule form.
- The parallel management of many states.

An interactive graphic interface is available, it allows users to access, define, test and modify the expert system objects and rules. The knowledge representation formalisms are : classes, objects, methods, inference rules and tasks.

Smeci can be foreseen as a multiformalism programming environment, it enables user to define rules, classes, methods, objects, etc. We can also consider it as an object-oriented system integrating the notions of objects or instances, class hierarchies, inheritance rules, ...

2.2 Centaur

Centaur [Baa87] [INR89] allows to define formalisms in the form of concrete and abstract syntax specifications in Metal [KMM83], pretty-printer in PPML and natural semantics in Typol [Des88]. It is a generic interactive environment. When given the formal specification of a particular programming language -including syntax and semantics- it produces a language specific environment.

The specifications of concrete and abstract syntax are written in Metal. A Metal specification is a collection of grammar rules, with functions that specify what abstract syntax tree should be built. Pretty-printing of abstract trees is defined in the PPML formalism. A PPML specification is a collection of unparsing rules. The semantics is specified with a formalism called Natural Semantics in Typol. A Typol program is a collection of axioms and inference rules.

3 Why implementing Smeci in Centaur?

Dealing with Smeci languages: rule, class and method languages, consists in parsing, type-checking and compiling them. In Smeci, the parsing step has been performed in Yacc

[Joh79], and the type-checking one has been realized by message passing. All these works have been done "*by hand*". It's why this project has been proposed. After having dealt with Smeci languages, we were wondering if there shouldn't be an easier way to realize this work automatically by a well known tool.

Nowadays, there exists systems which generate interactive programming environments to define formalisms or languages. These systems provide environments in which a programmer can design, implement, edit, correct, test, parse, control, validate and maintain his programs. Examples of these systems: Centaur, Mentor and Cornell. Why do we not exploit them ?

4 Application overview

From Smeci system integrating task, rule base, class, object, method, slot, state tree and reasoning notions, we have only considered the rule and class languages simplified in their structure. This reduced system is called μ-**Smeci.**

A μ-Smeci rule is defined by its name, its declarations, its premisses and its conclusions. A μ-Smeci class or category is defined by its name, its super-class and its slot list which can be empty. A μ-Smeci slot is described by its name, its type and its value. A type system has been built for the category slots. The syntax and semantics of these formalisms have been defined. Thus, the μ-Smeci languages have been treated in Centaur.

5 Application examples

5.1 μ-Smeci rule

```
defrule   weather
let       x a thermometer
```

```
                y a season
if              degree.x = 35
then            nature.x = summer
endrule
```

Thermometer and season are two categories, the notation degree.x denotes an access to the degree slot of the x object, and nature.y denotes an access to the nature slot of the y object. defrule, let, if, then and endrule represent the μ-Smeci rule keywords.

5.2 μ-Smeci class

```
category  Port  of mother  Object :
          Setting : object Setting ;
          Dike : list object Dike ;
          Agitation : object Agitation ;
          length : interval 0 .. 100 ;
```

In Smeci, every class has only one super-class called mother category. A list of slots is given to show their syntax. The slot types are defined in the Metal specification, a slot can have no value.

5.3 μ-Smeci rule language in Metal

As it is explained above, a Metal specification for the μ-Smeci rule language RL, includes:

- A *concrete syntax* for RL:

  This consists in BNF-like rules which describe the textual representations of legal

programs. These rules will be analysed by a parser generator (Yacc) to construct a parser for RL.

```
rules
   <defrule> ::= "defrule" <ident>
                 "let"     <dec_s>
                 "if"      <exp>
                 "then"    <conc_s>
                 "endrule" ;
```

- An *abstract syntax* for RL:

  The abstract syntax defines the structure of internal arborescent representations of programs. It is made of *operators* -corresponding to the nodes of the trees representing programs- and *phyla* or types of these operators.

```
abstract syntax
   defrule   ->   IDENT DEC_S EXP CONC_S ;

   IDENT     ::=  ident ;
   DEC_S     ::=  dec_s ;
   EXP       ::=  and  or  equal  access ;
   CONC_S    ::=  conc_s ;
```

- A collection of tree building functions:

  These functions are responsible of building trees when the BNF-like rule is encountered during parsing. They provide the connection between the concrete and the abstract syntax of RL.

```
defrule(<ident>,<dec_s>,<exp>,<conc_s>)
```

<ident>, <dec_s>, <exp> and <conc_s> are sub-trees of the tree labelled by the defrule operator.

5.4 Unparsing of μ-Smeci rule language in PPML

The developed Metal specification is used to define the μ-Smeci rule language syntax and to produce abstract trees. The PPML formalism will be used to associate concrete layout representation to abstract structures. So, a pretty-printing specification is a set of rules, where each rule expresses a translation of an abstract syntax tree into a pretty-printing language.

```
defrule(*ident, *decs, *exp, *concs) ->

        [<v> [<hv>  "defrule" *ident]
             [<hv>  "let"     *decs ]
             [<hv>  "if"      *exp  ]
             [<hv>  "then"    *concs]
             [<hv>  "endrule"       ]
        ];
```

Where the left-hand side matches any tree labelled by the defrule operator and having the same arity, while the right-hand side corresponds to a pretty-printing format specified in a *box language*. The same work has been done for the μ-Smeci class language: the syntax in Metal and the pretty-printer in PPML.

5.5 μ-Smeci class language in Metal

The same approach has been adopted to define the μ-Smeci class language in Centaur. So, we start by defining the concrete syntax specifications, then the abstract syntax ones. We give an example of the category definition:

```
rules

  (1)  <defcateg>  ::=  <categ_with_slots> ;
         <categ_with_slots>

  (2)  <defcateg>  ::=  <categ_simple> ;
         <categ_simple>

  (3)  <categ_with_slots> ::= "category" <ident>
                              "of" "mother" <ident> ":"
                              <slots_s> ";" ;
         <defcateg>(<ident>.1,<ident>.2,<slots_s>)

  (4)  <categ_simple>     ::= "category" <ident>
                              "of" "mother" <ident> ";" ;
         <defcateg(<ident>.1,<ident>.2,slots_s-list(()))
```

In this example, we show the BNF-like rules required for defining the category syntax followed by the tree building functions. In the first and second rules, we express that a category may have an empty or full list of slots. In the third and fourth rules, we reveal the syntax of the

syntax of the two cases, and we build the corresponding abstract trees.

The operator definitions and their declarations are written in the abstract syntax part of the Metal specification as follows:

```
abstract syntax

    defcateg   ->  IDENT IDENT SLOTS_S ;
    slots_s    ->  DEFSLOT * ... ;



    IDENT      ::=  ident ;
    SLOTS_S    ::=  slots_s ;
    DEFSLOT    ::=  defslot ;
```

The lowercase identifiers represent the operators or the abstract tree nodes, while the uppercase ones are the types or phyla of these operators. `defcateg` is a ternary node, accepting an `IDENT`ifier as a first and second son, and a list of `SLOTS_S` as a third son. `DEFSLOT * ...` denotes a star list which can have any number of elements including none. All these phyla must be respected in the semantics definition in Typol by the *judgement* notion, which gives the structure and the type of a sequent in an inference rule.

5.6 Unparsing of μ-Smeci class language in PPML

The abstract syntax trees built in the Metal specification must be unparsed in PPML. We give an example of pretty-printing a μ-Smeci class as it has been mentioned in the μ-Smeci rule language:

```
rules

 (1)  defcateg(*name,*mother,slots_s()) ->

        [<h> "category" *name "of mother" *mother ";"] ;


 (2)  defcateg(*name,*mother,*defslot) ->

        [<v> [<h> "category" *name "of mother" *mother ":"]
             [<hv> *defslot] ";" ]
```

The first rule gives the format of a category definition without slots and the second one gives the format of a full list of slots. The formats are given in *box language* where <h> denotes an horizontal separator, <v> a vertical separator and <hv> an horizontal separator with an automatic vertical folding applied when the horizontal composition does not fit into the width of the associated output device.

5.7 Type-check of the rule language in Typol

We present an example of inference rule developed to implement the type-check of μ-Smeci rules with respect to class language in Centaur. We have checked variable declarations, types, slot existence and assignments in a μ-Smeci rule in relation to category definitions. We have implemented a simple semantics of slot inheritance in the classes.

```
dec  (env_s[], envclasses |- decs  : envvar) &
prem (envvar , envclasse  |- pred  : type  ) &
conc (envvar , envclasse  |- concs : concs )
-----------------------
envclasse  |- defrule(id, decs, pred, concs) ;

                .
                .
                .

set dec is
    <body>
end dec ;
```

In this example, we can notice the structure of a Typol *inference rule*. An inference rule has two parts: *premisses* and *conclusion*, separated by an horizontal line. A conclusion is a sequent and premisses are a list of sequents separated by ampersands. The rule explains when a μ-Smeci rule is well typed: if given an empty environment of declarations and an environment of classes (defined in Metal), the declarative part of the μ-Smeci rule produces a new environment called `envvar`. From this environment, we type-check the premisses and

the conclusion. A **set** is a named collection of inference rules. This collection of rules is a complete formal system, it gives to Typol program the modularity characteristic.

Finally, it is important to notice that all the specification languages manipulated by Centaur have as a common feature the use of abstract synatx operators. It is the system responsibility to adapt them to different tools.

6 Studied problems

The definition of Smeci languages in different Centaur specification formats is feasible. But some difficulties have been faced in modeling class hierarchies, type-checking inference rules with respect to these classes and dealing with inheritance.

Simple inheritance in Smeci is developed in Typol in a static way: value, method and instance inheritance are not dealt with. In order to see Centaur's behaviour towards *multiple inheritance*, we have studied many systems which integrate it [FP90] [Dug86] [HO86] [BI82] [Hl86]. After study, we were able to deduce a solution to deal with it by Typol inference rules: it's the **class precedence list** mechanism of Clos [Kee89] which allows to degrade multiple inheritance to simple one [DH88].

6.1 Class precedence list principle

Clos calculates the *class precedence list* of each class, it is a total ordering on the set of the given class and its superclasses. The total ordering is expressed as a list ordered from most specific to least specific. The element order in a class precedence list must respect the two following rules:

- *Each class has priority over their superclasses.*

- *The local precedence order, which is a list consisting of the class followed by its direct superclasses in the order mentioned in the defining form, must be respected.*

Clos starts by the class definition, applies the two rules and gets a set of constraints in a local level. From this step, it applies rules on the direct superclass definitions, then on their superclasses. This process is repeated until the root class is encountered. The result is an *ordering constraint set over the classes.* The next step consists in finding a total ordering on the class precedence list satisfying all the ordering constraints. Clos proceeds by *Topological Sorting* [BDG+88] of this set of constraints.

7 Evaluation

The possibility of programming a class precedence list algorithm in Typol, presents a solution for modeling multiple inheritance in Centaur but makes Typol losing its natural semantics implementation characteristic. So, we hardly express procedural programming concepts by a great number of declarative rules.

For simple inheritance in Smeci, we dealt with the slot inheritance of one class by its subclass, the disadvantage is that we have to compute inherited slot list at each time it is used.

From this work, a syntactic editor for rule language, a syntactic editor for class language and a type-checker ensuring the validity of rules in relation to class definitions, are issued. This result could make up a subset of Smeci system, in which we could edit and control programs. It can be illustrated in the following figure:

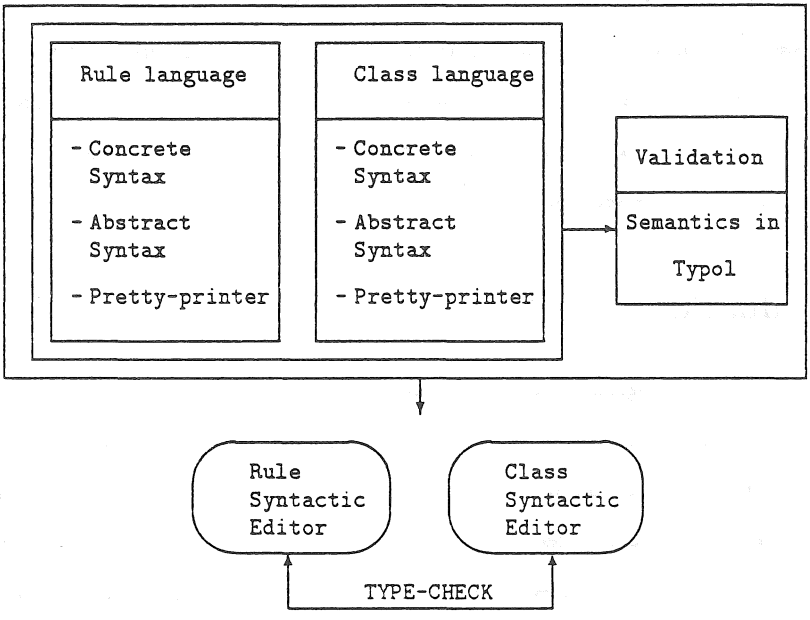


Figure 1: Application representation and issues

8 Conclusion

This work concerns the designers of multiformalism object-oriented systems in Artificial Intelligence or others. Its purposes are as follows:

- It's interesting to know the existence of tools able to realize many tasks in an easier and more flexible way.
- It's interesting to see the behaviour of Centaur, which is developed for classic language design, with some particular systems manipulating several languages at the same time and consisting of object-oriented programming notions: classes, instances and inheritance.
- Object-oriented programming tends to be more and more used, so we will need to design a lot of languages of this generation. Therefore, testing Centaur behaviour with these formalisms is interesting for object-oriented approach as well as artificial intelligence field which uses these new concepts.

Acknowledgements

This paper would not have been possible without the help of the Secoia team members led by B. Neveu and in particular the assistance of **Olivier Corby** enriched by numerous discussions and advice. I also want to acknowledge C. Fornarino for his useful remarks related to this work.

References

[Baa87] P. Borras and al. Centaur : the system. Rapport de recherche RR-777, INRIA, décembre 1987.

[BDG+88] D. G. Bobrow, L. G. DeMichiel, R. P. Gabriel, S. E. Keene, G. Kiczales, and D. A. Moon. *Common Lisp Object System Specification.* ISO/IEC, February 1988.

[BI82] A. H. Borning and D. H. H. Ingalls. Multiple inheritance in smalltalk-80. In *The National Conference on AI*, pages 234–237, Pittsburg, August 1982.

[Des88] T. Despeyroux. Typol: A formalism to implement natural semantics. Rapport de recherche 94, INRIA, Mars 1988.

[DH88] R. Ducournau and M. Habib. La multiplicité de l'héritage dans les langages à objets. Rapport de recherche 87/1, Université de Bretagne occidentale, Juin 1988.

[Dug86] P. Dugerdil. A propos des mécanismes d'héritage dans un langage orienté objet. *CIIAM*, pages 67–77, 1986.

[FP90] M. Fornarino and A.M. Pinna. *Un modèle objet logique et relationnel: le langage OTHELO.* PhD thesis, Université de Nice, Avril 1990.

[Hl86] J. Hendler. Enhancement for multiple-inheritance. *Sigplan Notices 21(10) ACM*, pages 98–107, October 1986.

[HO86] D. C. Halbert and P. D. O'Brien. Using types and inheritance in object-oriented languages. Rapport de recherche, Digital Equipment Corporation Object-Based Systems Group., Avril 1986.

[II89] INRIA and Ilog. *SMECI : Le manuel de référence.* Gentilly, 1989.

[INR89] INRIA. *The CENTAUR User's Manual.* Sophia-Antipolis, 1989.

[Joh79] S. Johnson. *YACC : Yet Another Compiler Compiler.* Bell Laboratories, 1979.

[Kee89] S. E. Keene. *Object-Oriented Programming in COMMON LISP.* Addison-Wesley Publishing Company, 1989.

[KMM83] G. Kahn, B. Mélèse, and E. Morcos. Metal: A formalism to specify formalisms. *Science of Computer Programming 3 North-Holland*, pages 151–188, 1983.

[Lej90] W. Lejouad. Environnement de programmation pour générateur de systèmes experts: Smeci sous centaur. Rapport de dea, INRIA - Sophia Antipolis, Septembre 1990.

[MNC+89] G. Masini, A. Napoli, D. Colnet, D. Léonard, and K. Tombre. *Les langages à objets.* InterEditions, 1989.

Communicating Petri Nets

G. Degli Antoni[1]

Abstract

In this present paper basic Petri Nets models are re-viewed. Their difficulties in modelling some real systems (cognitive and epistemological difficulties) are analyzed. In order to overcome these difficulties, the basic models (Condition Event/Elementary Net) are extended. The proposed extension implies a basic mechanism of communication, whereby the holding of a transmitting condition lets the receiving events to fire, whenever their have concession in the classical sense. The corresponding token rule is suggested, leading itself to a natural extension of the notion of Petri Nets, where there is a communication between different Nets (components) modelling parts of a single system.
Simple examples are given in order to illustrate the easiness in modelling the proposed extension of Petri Nets. It is observed that the proposed extensions fit with a general movement towards object orientation in modelling complex concurrent systems.

1. Introduction

Petri Nets are one of the most interesting formalism to model systems. A large number of people has considered them as an important subject of theoretical research. Certainly the nets ability to nicely represent the notions related to causality and concurrency makes Petri Nets a field where fundamental research could lead to a deeper understanding of processes and systems.

[1] Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano
Via Giuseppe Colombo 46, 20133 Milano, e.mail degliantoni@hermes.unimi.it

The research has been developed in the frame of a cooperation project (Personal University Project) between Milano University (Italy) and Catholic University of Asunciòn (Paraguay), sponsored by the Minister of Foreign Affairs jointly with the Minister of University and Scientific Research of the Italian Governement.
In the PU project the the Communicating Petri Nets are related to an hypermedia modelling system which is under developement by S. Aranda, M. Bondi, G. Grossi.

Nevertheless, those that have used Petri Nets for practical purposes realize difficulties in representing real life systems and processes. These difficulties are well predicted in Petri's conferences on Nets. In fact, frequently Petri claims that in every field the right language has to be used. Petri's position is extremely honest and certainly correct, but it is difficult to avoid a positive thinking that suggests to make research in order to introduce small modifications in Nets to improve their expressiveness, and their usability.

That research would be of a purely empirical nature as it happens exactly any other really scientific discipline. Only human non mathematical judgements can justify sentences of any theory axiomatizing some notions (sets, numbers, spaces, space time...). Obviously, given any axiomatization, its consequences can be subjected to the mathematical investigation, just to prove theorems or to enlight the empirical nature (models, both mathematical or cognitive ones) of the concept captured by the axioms.

In the case of Petri Nets, the basic definitions collect fundamental ideas: their capability of representing extremely wide classes of processes makes them an interesting tool for practical or for epistemological research, i. e. for making science through modelling activity.

It follows that the choice of good models for Petri Nets is an important matter.

Perhaps a research channel has to be opened, whereby comparison of modelling power of net definitions have to be performed in order to eliminate all the doubts that some practitioners maintain regarding the usability of Petri Nets.

The loss of literature and the extreme differences in formalism used to reason about Petri Nets, together with mathematical difficulties of many papers, is transforming the field of Petri Nets in a purely academic subject in which it is honestly difficult to have clear answers to simple problems.

The state of affairs suggests that a non completly mature basic definition of Petri Nets has been reached and that we have to look again at the first papers with curiosity and criticism, even considering the many researches that have been developed since the early days of Petri Nets.

Following this esprit the present paper suggests a new class of Nets derived from Petri's ideas in order to overcome some of the difficulties encountered in using Nets. The results seem interesting for various reasons, even related to the nature of nets coming out from our analysis. The resulting nets, that we will call *Communicating Petri Nets* (CPN), certainly can be classified as Object Oriented Petri Nets. It is interesting to note that our result is coming out not as a consequence of the actual pressure towards object orientation, but rather from the aim of understanding net behaviour in some critical situations (1).

2. A concise re-view on net definitions

A concise review on nets and particularly on their token games is here introduced for sake of synthesis and to build up with the reader a common view on nets. Generally, in the many models of net behaviour specification a token game is considered, eventually described through some mathematical definition that gives a better insight on the meaning of net behaviour. Token games bring a lot of intuition on problems related to the net behaviour through the physical nature of the token game itself.
Generally the various models of token games used in literature make transition firing (event occurrence) change the tokens pattern in the places directly related to the considered transitions. The set of holding conditions (case) changes as due to transition firings (processes). Firing is subjected to some limitation that are generally called concession conditions. Concession holds for sets of transitions. Concessed transitions can occur concurrently if conflicts are not present. A concessed transition is in conflict with another one if its firing changes the token pattern in such a way that it is impossible for the second one to fire. Generally a conflict is a symmetric relation. The interaction between conflicts and concurrency give rise to interesting confusion situations where conflicts avoidance is an observer related property (subjectivity). Class of nets depends heavily on the choice of concession and firing rules and on the nature of the places: these can be made holding one or more tokens. Extensions have been obtained considering structured tokens. Concession and firing rules will be related to those data structures associated to the token types.
Properties of class of nets have been studied extensively in order to relate local behaviour to the general one and static properties to dynamical ones. In this latter category for our purposes we want to recall safety: this property tries to identify dangerous situations in real systems, if there are contacts in the corresponding net model.
In order to simplify the burden of so many possibilities, Petri has introduced Net Morphisms, suggesting that all net models should be reduced - through net morphism - to some elementary net model (2). The preferred Petri's choice has been the Condition Event (CE) nets model. Elementary Net Model is sometimes preferred (3). In these models places hold a single token and are called conditions. Transitions are called events and have concession if all their input conditions (places) and no output conditions hold. The occurrence of an event changes all its input and output conditions.
The concise view given here can be easily translated into a more compact mathematical notation with the danger of loosing some underlying idea. We prefer to leave it as informal and as concise as possible, preferring possible ambiguities (if any) to the difficulty in reading formal definitions. These are certainly needed when properties have to be studied, not now in our definitional approach.

3. A cognitive difficulty with Petri Nets

In all models of Petri Nets, occurrence of events, that is firing of transitions, is essentially a change of cases, that is a change on the state of the token =holding places (conditions). Some condition will cease to hold while others will begin. A fundamental idea is indeed captured: no event occurs anywhere without change on some condition (extensionality).

The unfortunate state of affairs with current definitions is that some extremely simple system can not be represented. To exemplify, let's consider the flowing of current after a switch has been closed in an electrical circuit or any other analog system. The way to represent that system is to consider the concurrent changes of both switch and current as events (Fig. 1).

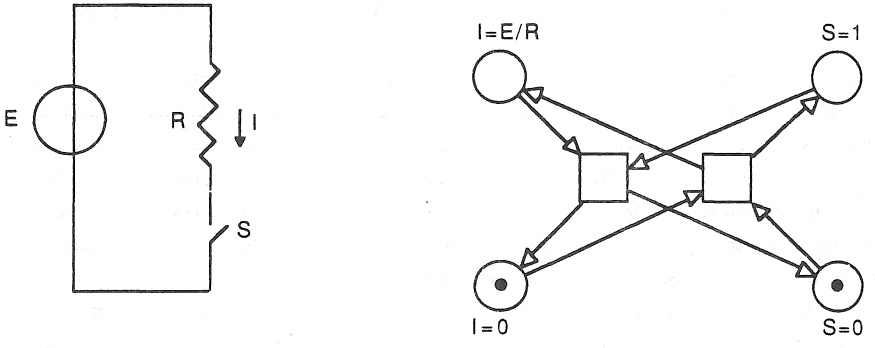


Fig. 1

It is acceptable that there must exist some form of concurrency between the changing of the state of the switch and the change of current flowing through the switch itself. But the intuitive causal dependence of the current flow consequent tp the switch closure has been lost. But it is completely unacceptable to let the transition of the switch be dependent on the state of the current flow. In some sense the net brings some information on the phenomenon: infact it is quite likely that at some level of description the state of the switch is related to the current flowing on it. So the net models only a very limited state of affairs, perhaps acceptable in some extremely deep sense. But the simple intuitive meaning of the circuit fully understandable by any human mind is lost, indicating some fundamental difficulty. The fenomenology implied in the example shown before is perhaps related to some deep discrepancy between composition rules in nets and composition rules in circuits. If these considerations are correct, it will be probably difficult to understand a better compositionality in nets working inside the net theory as actually developed. Unfortunately, the example shown has to do with all

systems where changes in components modify other components. The state of affairs related to the selected example has consequences in all real situations. If the discussion is correct, then it is not difficult to understand why nets are used and then abandoned after their strong maieutical power has been depicted to analyze the system at hands.
With respect to the latter assertion, it is certainly likely that nets owe their maieutical power to many reasons, certainly related to the double role of places. Infact places in nets represent an indication act on the world (4), that is some entity and its model as assumed by the reader (or modeler). Moreover, the effort that modelling activity requires, is probably responsible for the judgement on the maieutical power of nets. Maieutical power and simplicity of the models have made Petri Nets the most known cognitive tool for the representation of real systems behaviour with concurrence. The difficulties reported in this chapter do not weaken the Petri Nets as a modelling tool. Viceversa, we think that all criticism will improve a better understanding of nets and their applications to real life problems.

4. An epistemological difficulty

The last chapter has discussed some difficulties on modelling systems with Petri Nets in their actual model. The nature of the difficulties discussed are related to real systems and their models, a topic rarely analyzed in the theoretical Petri Nets literature. We call those difficulties cognitive ones. There are others difficulties with Petri Nets when their interaction with agents (for example the reader using them for simulation purposes it is taken into account).
The problem can perhaps be classified as an epistemological one and it is related to the nature of the observation on nets. They are done just as if the agent using nets does not influence the net behaviour. The agent can use information on nets as a system with complete information on everything is happening on the net. For example, he can decide to make event occur (transition firing) taking the information from its own state, without discussing on how this happens.
This implies some form of communication from the net to the observer and from the observer to the net. For example, if simulation is made for comparing a model A with a model B, an event on A is made to occur from the analyst just considering some state in B in order to compare consequences of changes. More precisely, if some state of affairs is present in one of the system, an event will be made to occur in the other one.
It is a custom of all sciences to consider abstractions. In our example we have to abstract from the existence of the observer but we have to take into account what happens.

What happens can be understood using net concepts if we consider that some event can occur if and only if there is an *external condition of occurrency,* as it happens when by hand simulation we decide to fire a transition. That decision does not necessarily modify the state of the affairs that has led to the firing decision.
A similar state of affairs will hold in communication from one net model to other net models.

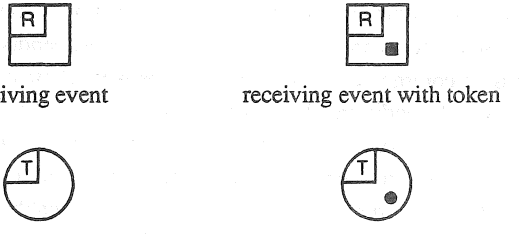
5. Extending the net model

Extending the actual models of net behaviour can help solving the epistemological and the cognitive difficulties discussed above.
The discussions above imply that those difficulties are not present if some form of state is associated to the events and that state depends on what happens. For example, the current can flow through a switch only if this has been closed. In order to take into account the suggestion provided by the present discussion, an extension to the net re-view presented above is convenient. We will call the resulting nets Communicating Petri Nets.
Communicating Petri Nets are a set of Condition Event (or Elementary) Nets that will be called Components. Each of these Components behave exactly as usually and as described in our re-view for elementary net models.
A Communication Structure is superimposed to the components. The Communication structure is given as a set of pairs *(t, r)* called elementary channels. The first element of each pair is a condition of some Component, the second element is an event of a different Component. So a pair *(t,r)* indicates an elementary channel whereby the holding of a transmitting condition *t* sends some signal to the corresponding receiving event *r*. A component A communicates with a component B $(A \neq B)$ in a Communicating Petri Net if there is a set of elementary channels with transmitters in A and the corresponding receivers in B.
We will call the events that are receivers of some condition R-Events (in word: committed events). Transmitting conditions will be indicated with T-Conditions. In a pair *(t,r)* we will call *r* committed by *t*. It is to be noted that a transmitter *t* may commits different receivers, and that a receiver can receive commitments from distinct transmitters.
To complete the model, concession and firing rules must be extended. There are many possible ways to do it. We suggest here one possibility that agrees fully with Petri's attitude regarding timing details, i.e. timing are fully excluded from definitions. Signal reception by an r-event is modelled putting a token in it . The sending of messages by a t-condition *p*, in our preferred choice, happens when conditions begin to hold, or equivalently when an event (committed or not) has placed a token on *p*.

A choice has still to be done: to decide how a c-event stores the received message (a token) from some t-condition. Our choice is the following: if a c-event has concession in the spirit of the standard concession, then the received message will be stored in it just to the firing of the event itself.
Now we can completely formulate our new c-concession and c-firing rules. A standard event (one that is not an R-Event) has c-concession if it has concession in the usual well known sense. An r-event has c-concession if it has concession in the usual sense and it holds a token (commitment) in it. Events occur (transitions c-fire) only if they have c-concession c-firing of an event is standard if that event is standard. If the event *e* is an r-event, then the token game proceeds as usual but, a token (commitment) is placed in every uncommitted (that is a non holding token) standardly concessed receiver event associated to the transmitting output conditions of the considered event *e*. It is to be noted that in these definitions a situation similar to the contact is not considered.
As any other net, our Communicating Nets require an initial marking. The marking can be associated to both conditions and events. It is easy to accept as initial marking definition one that requires only initial marking on places. Non receiving events are just those events that model some autonomy in our model. It will be their responsibility to send tokens to the transmitting conditions, making possible a message sending from t-conditions to r-events sites. Side effects of the event occurrence is the reception of commitments by receiving events concessed in the usual sense.

6. The notation for the extended model

The basic elements of the notation are shown in fig. 2. The notation is suitable for further extensions of the model in order to consider (for example) data types associated with conditions and the nature of messages. In our simple model, the letters T in places and R in transitions are redundant. Elementary communicating channels are described by the correspondence between symbols indicated at the t-conditions and the corresponding r-events, for example using capital letter in the t-condiitions and the lower case in the corresponding r-events.



receiving event — receiving event with token

sending condition — sending condition with token

Fig. 2

7. Discussion of the extended net model

The proposed net model can be considered from many points of view. But perhaps the most striking aspect is the introduction of tokens in committed events. This introduction has some pleasant consequence: messages became side effects, simple causality relations are naturally modelled. Moreover, some form of independence has been modelled between the behaviour of transmitter and receiver systems as well as a form of dependence among them. Our choice makes the sending of a message possible at any time even if the preceeding ones have not been consumed on the same elementary channels. The presence of tokens in some sense only indicated that the communication is used instrumentally and has necessarily some form of memory (messages travelling).

8. System analysis, design and implementation with Communicating Petri Nets

Analyzing systems with our model is a rather simple matter. Components have to be identified. These have to be modelled as CE o EN nets, or with any similar formalism. Initial marking has to be given. The Communication Structure has then to be identified by listing the elementary channels. Simulation or other knowledge elicitation means has now to be used in order to verify the correctness of the model, for example comparing his mathematical properties with abstractions of the real system.
The identification of the components in an analytical work could be a matter of taste. Where the system is build of separately described objects, then the best policy it is probably to consider these as parts. In these cases choices are less arbitrary and results more reliable. It is to be noted nevertheless that a good analysis of a systems does not introduce arbitrary choices, but possibly classes of behaviour.
For example an elevator is made of a cabin, doors, buttons with various purposes , lights, a platform to detect presence of people in the elevator, a control unit and some users. Each of these objects can be described with nets (Appendix). And the overall behaviour depends on communication (interactions) between these components.
The cabine has a set of conditions describing its state (i. e., where it is). Transitions from the various states are possible and are due to elementary messages coming from a centralized control unit or from decentralized components entering in suitable states. Doors (one for each state of the elevator) will be blocked, openable, open. There will be buttons to call the elevator outside the doors. Others will be used to select the destination place. Buttons are a two-state entity. Light are used for various purposes and are displaced inside and outside the elevator. Outside the elevator the light will be located at each starting and destination place. The state of the platform indicates if people is inside the cabine. People will require services identified by the starting and

destination places. The control-unit will eventually centralize requests from the starting places and will send messages to the components in order to make them change their state according a good policy. The elevator will be correct if it will correspond to the real system modelled. In the case of a design the elevator model will be correct if it will model the requested services: these can be described by a couple of locations: the starting and arrival location of all the users acting concurrently.
Having obtained a good specification of a concurrent system the problem of *implementing* it is related to the nature of components used to implement the system considered. In fig. 3 a Set-Reset Flip Flop is modelled. In fig. 3.1 and 3.2 transitions due to an input signal are shown. In fig. 3.4 the flip obtained using standard switching synthesis procedures is shown. It is outside the objective of this paper to suggest general implementation methodologies starting from a net description. Nevertheless, something can be said quite easily. In the case some component is naturally implemented (the cabin....) nothing has to be done. The control unit must be synthesized, for example using digital circuits with the approaches implied in fig. 2.. It is a matter of activity partially feasible by automated processing the problem of building some circuits or software realization of the control unit itself starting from the full specification of the entire system with CP Nets The reader can exercize itself implementing a lift as modeled in fig. 4 and fig. 5.

9. Object orientation in Communicating Petri Nets

Real system are frequently made by components that can be described in more general common terms. The term class applies naturally to these cases. Reasoning with our nets, the classes are nets (communicating Petri Nets or other nets). Actual instantiation of classes gives rise to actual objects that in our terms correspond, for example, to components. Actual object behave according to their methods (the net description). Messages are sent through objects.
The inheritance relation among classes and actual objects requires a bit of attention in the case of nets: infact such a relation can be modelled in terms of morphisms!
What we have seen in the present paper strongly supports the evidence that object orientation is particularly suitable to improve the expressiveness of Petri Nets as well as for improving the possibility of using Petri Net theoretical concept for a deeper understanding of object orientation in presence of concurrency.
We hope that this paper has given a first insight in such an interesting matter.

Acknowledgements

Sincere thanks are due to Angelo Beltramini, Anatol Holt, Giorgio De Michelis and Astasia Pagnoni for our many discussions about Petri Nets.

References

1. A. Beltramini, G. Degli Antoni, P. Perego,
"Petri nets and Switching Circuits, In Petri Nets and Related System Models", Newsletter 18, October 1984. Published by Gesellschaft fur Informatik, Bonn.

2. C.A. Petri
"Concurrency as a Basis of Systems Thinking. In Finn V. Jensen; Brian H. Mayoh, Karen K. Moller (Eds): Proceedings from the 5th Scandinavian Logic Symposium, Aalborg, January 1979.

3 P.S. Thiagarajan,
"Elementary net systems". In W. Brauer, W. Reisig and G. Rozenberg (Eds.): Petri Nets: Central Models and Their Properties, Advanced in Petri Nets 1986 Part I, Lecture Notes in Computer Scienze (Vol. 254). Berlin: Springer Verlag, 1987.

4 Spencer Brown
"The Laws of Form",
Pinguin Book

Note added in Proof
The author is aware of efforts directed to introduce object orientation in Petri Ne modelling by product of this note. Some of them is reported here.

- Y. Chongyi Towards a general solution of system automation: A net specification of a N_Lift System. GMD, St. Augustin 1, West Germany, January 1990.

- C. Ghezzi, D. Mandrioli On Eclecticism in Specification: A Case Study Centered Around Petri Nets, Techical report 87 - 008, Politecnico Milano.

- D. Buchs, N. Guelfi CO-OPN: A Concurrent Object Oriented Petri Nets Approach for System Specification, internal report LRI n°616, UPS Orsay, 1990.

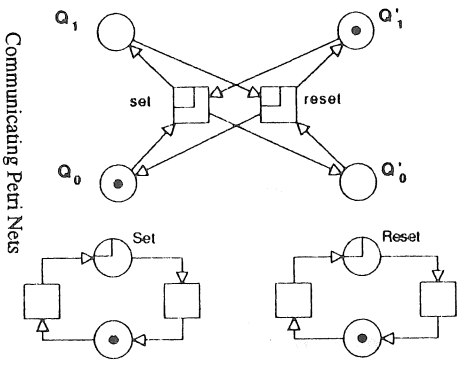


Fig. 3.1

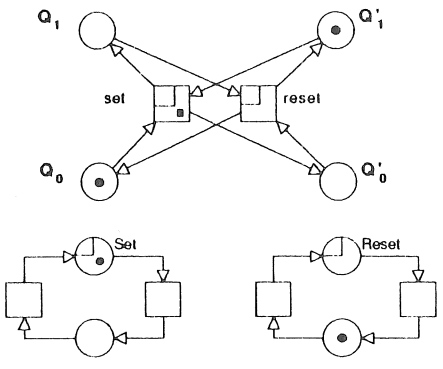


Fig. 3.2

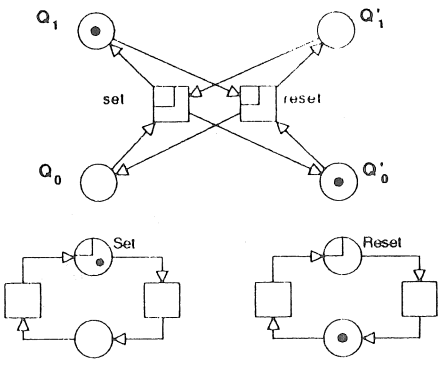


Fig. 3.3

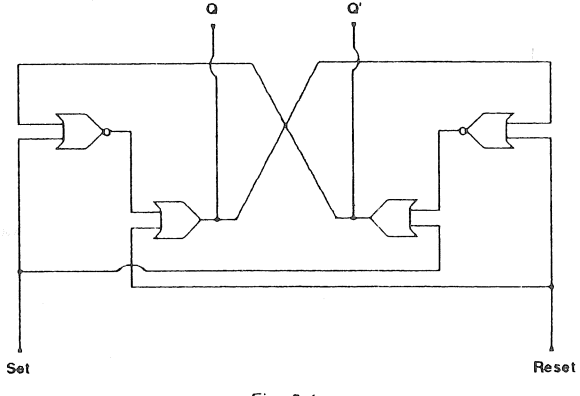


Fig. 3.4

Fig. 3

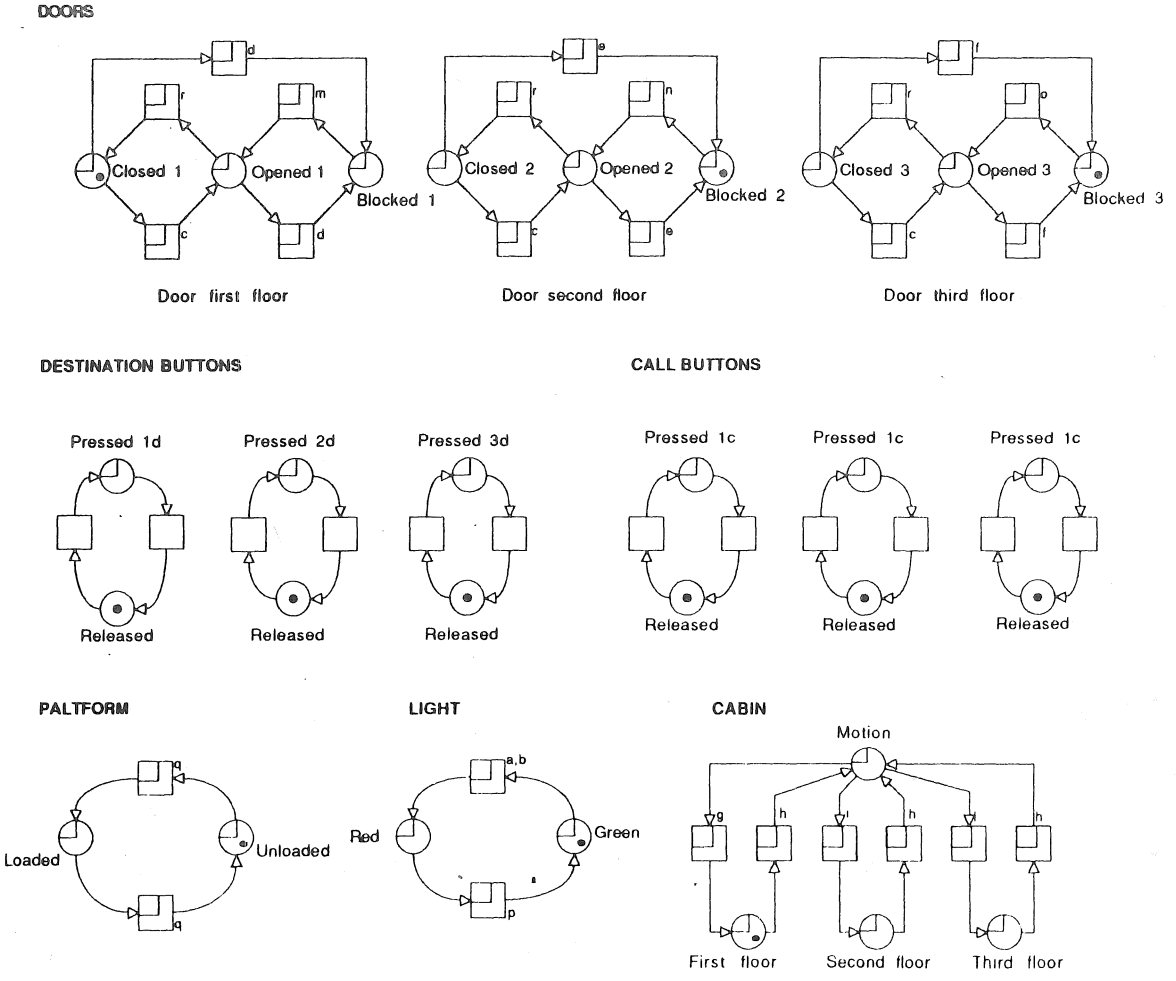
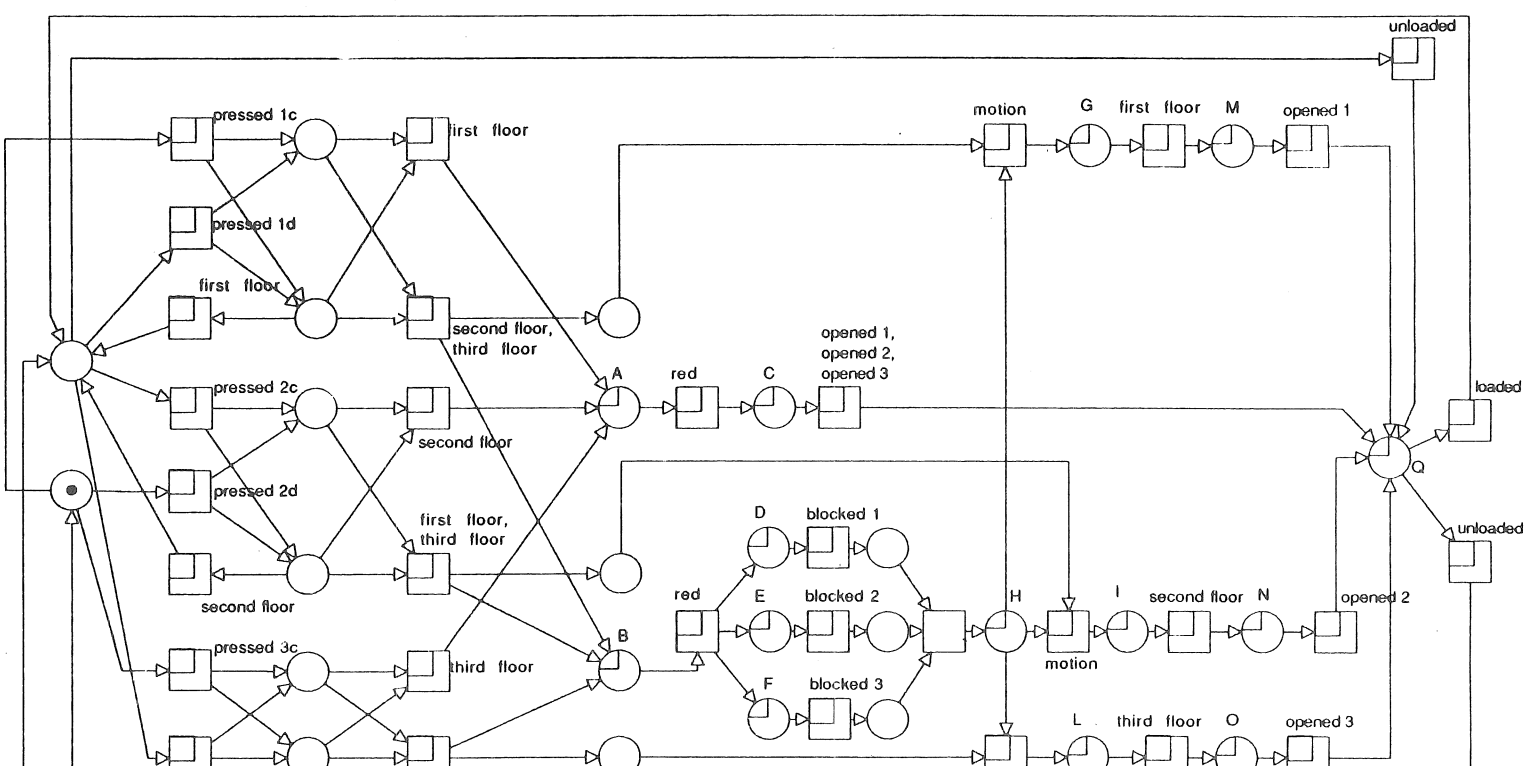


Fig 4

A CPN model for the lift



CPN model for the Control Unit of the lift

Fig. 5

Un estudio de técnicas de estructuración de software: aplicación en el ambiente Adele

N. Belkhatir R. Casallas W. L. Melo
Laboratoire de Génie Informatique
BP 53X
38.041 Grenoble Cedex France
e-mail: casallas@imag.imag.fr
phone: +(33) 76 51 46 68

Resumen

Este artículo presenta un estado del arte sobre la utilización de técnicas de clasificación para el mantenimiento de software de gran tamaño. Estas técnicas permiten hacer reagrupamientos sobre un conjunto de objetos atómicos. Para hacer esto, muchas de ellas se apoyan sobre el domino de la taxonomía numérica y sobre técnicas de inteligencia artificial. Nosotros presentamos las caracteristícas principales de las técnicas de clasificación numérica y conceptual. Mostramos cinco ejemplos de herramientas que utilizan este enfoque. Las dos primeras utilizan las técnicas llamadas numéricas: la primera para encontrar las partes de un software suceptible de contener errores, la segunda, para proponer un reordenamiento para la integración del software modificado. Las tres siguientes utilizan las técnicas llamadas conceptuales. Su propósito es, respectivamente, encontrar la estructura lógica del software viejo, optimizar los procesos de reutilización y generar automáticamente bibliotecas de códigos reutilizables. Después de una comparación de estas diferentes técnicas, mostramos como nosotros podemos utilizarlas dentro del ambiente Adele. Terminamos por una evaluación sucinta de experimentos realizados.

Palabras claves

Técnicas de clasificacíon, mantenimiento de software de gran tamaño, reutilización, reestructuración, taxonomía numérica, técnicas de clasificación conceptual, programación global, Adele, métricas.

1 Introducción

El costo de mantenimiento y evolución del software está en progresión constante. Numerosos estudios [Pressman87] situan, para el próximo decenio, el costo de la actividad de mantenimiento aproximadamente como en el 80% del costo total del software existente. De ahí la necesidad de proponer y de desarrollar nuevos métodos y herramientas para ayudar en la automatización de esta importante etapa del ciclo de vida del software. Este artículo propone enfoques para la reutilización y el mantenimiento de la estructura del software de gran tamaño que se basan en técnicas de clasificación. Estas técnicas permiten obtener una estructura del software que esta en servicio, a partir del código fuente.

El software voluminoso (superior a 10^6 lineas de código fuente) tienen una gran vida útil en comparación con el tiempo invertido en su desarrollo. Durante este periodo, las modificaciones efectuadas, cualquiera que sea su tipo, (correctivas, evolutivas, adaptativas) desestructuran progresivamente el software. Las herramientas que se basan sobre las técnicas de clasificación permiten a partir del código fuente:

- de una parte, extraer la estructura real del software,
- y de otra, comparar con la estructura inicial del diseño para señalar los cambios eventuales.

Estos cambios necesitan una interpretación, que puede ser automatizada, para detectar las eventuales incoherencias en la evolución.

Otra aplicación de las técnicas de clasificación es la que concierne con los aspectos de reutilización de componentes de software. Paro esto, se utilizan las técnicas de indexación automática para derivar las propiedades de los componentes a partir de su documentación.

El artículo presenta las principales técnicas, ilustradas en ejemplos de herramientas y utilizadas dentro del contexto de mantenimiento de software. La sección 2 introduce las técnicas de clasificación. Mostramos sus principales características y funciones. La sección 3 presenta cinco ejemplos de estas técnicas en la etapa de mantenimiento. La sección 4 presenta una evaluación comparativa de experimentos realizados. La sección 5 describe brevemente el ambiente Adele y la evaluación de los primeros experimentos llevados a cabo, los cuales aplican las técnicas de estructuración al software que Adele administra.

2 Las técnicas de clasificación

Las técnicas de clasificación tienen por objetivo organizar conjuntos de datos que representan por ejemplo: medidas, observaciones, eventos. Esto consiste en formar clases donde en cada una se agrupan los objetos que tienen una fuerte similitud. Después [Fischer86], un modelo general puede ser definido como sigue:

Dados: un conjunto de objetos O.

Objetivo: identificar las subclases $s_1, \ldots, s_i, \ldots, s_n$ (subconjuntos de O), tales que la similitud de los objetos de cada s_i tiende a ser maximizada mientras que la similitud de los objetos de subclases diferentes tiende a ser minimizadá. La colección de clases se llama una clasificación [1].

[1] La organizació de clases y su estructura interna cambian en función de la técnica escogida. Cada aplicación tiene una técnica apropiada

Hemos identificado dos tipos de técnicas de clasificación:

- las técnicas llamadas de taxonomía numérica [Everitt80,skodal63,skodal88]. También se encuentra la expresión clasificación numérica para denotar esta técnica.
- las técnicas llamadas de clasificación conceptual [2] [Michalski80] desarrolladas recientemente y más generales en comparación con las técnicas numéricas.

2.1 Taxonomía numérica

Las medidas utilizadas para identificar la similitud entre los objetos es de naturaleza numérica. El conjunto de atributos que caracterizan los objetos es limitado y definido a priori. La similitud entre los objetos es evaluada utilizando una matriz de similitud. Según [Hutchens85], una matriz de similitud D, para un conjunto ordenado P de n elementos, es una matriz cuadrada D[n,n] tal que:

$$D(i,j) \geq 0 \qquad 1 \leq i \leq n \qquad 1 \leq j \leq n$$
$$D(i,j) = D(j,i)$$
$$D(i,i) = 0$$

Según [Fisher86], las técnicas numéricas pueden ser estructuradas en:

- Técnicas de optimización.
  Estas técnicas intentan formar una cantidad óptima de K particiones sobre un conjunto de objetos O. O es dividido en K grupos mutuamente excluyentes. El valor de K es dado por el usuario.
- Técnicas jerárquicas.
  Estas técnicas intentan crear un árbol de clasificación binaria, llamado dendogramme, sobre un conjunto de objetos O. Las hojas representan los objetos atómicos y los nodos internos los grupos de objetos. Las técnicas jerárquicas pueden todavía ser divididas en técnicas llamadas de aglomeración y de división. Estas últimas construidas en forma de dendogramme de forma ascendente o descendente.
- Técnicas de recubrimiento [3].
  Estas técnicas intentan construir grupos de objetos no mutuamente excluyentes.

2.2 Técnica Conceptual

Las técnicas de clasificación conceptual se basan sobre los atributos cuantitativos (medidas numéricas) y cualitativos que describen las propiedades. El objetivo de una clasificación conceptual no es sólo identificar las clases, sino también, determinar la estructura conceptual implícita. Se define una clase por la enumeración de sus elementos y por un conjunto de propiedades que definen las relaciones entre sus miembros.

Un estudio mas detallado de técnicas de clasificación conceptual y de ejemplos de comparación con la taxonomía numérica son presentados en [Fisher86, Michalski83]. [Fisher86] compara varios sistemas de clasificación conceptual y [Michalski83] describe con mas detalle el sistema que se basa sobre la técnica Cluster/2.

[2] "Conceptual clustering"
[3] "Cluping"

3 Ejemplos de aplicación en el dominio de la ingenieria de software

En esta parte presentamos cinco ejemplos de herramientas que utilizan las técnicas de clasificación para ayudar en el mantenimiento de software de gran tamaño. Los dos primeros ejemplos utilizan la técnica de clasificación numérica para: (1) identificar las partes del software sujetas a error, (2) aportar una ayuda en la integración de software de gran tamaño. El tercer ejemplo utiliza una técnica de clasificación conceptual para generar nuevas estructuras jerárquicas del sistema o para hacer la comparación entre la estructura producida y la deseada. El cuarto y quinto ejemplo utilizan técnicas conceptuales diferentes para crear y mantener bibliotecas de código reutilizable.

Hemos escogido ejemplos de trabajos recientes y aplicables en la fase de mantenimiento. Los tres primeros ejemplos utilizan como único recurso el código fuente del software. Los dos últimos utilizan otros tipos de información como por ejemplo los índices derivados de la documentación por una herramienta de indexación automática.

Este trabajo describe experiencias de la aplicación de estas técnicas en el dominio de la ingenieria de software. No discutimos los algoritmos utilizados ya que estos estan suficientemente documentados (cf. bibliografía).

3.1 Identificación de las partes del software que contienen errores

[Selby88a y 88b] utiliza un técnica de clasificaión numérica para extraer del código fuente una estructura jerárquica del sistema. Los autores utilizán, como criterio de clasificación, medidas de interacción para un software distribuido en diversos tipos. La tabla 1 muestra algunos ejemplos de tipos de interacción definidos como sigue:

- Interacción potencial: está definida como una tupla (p,x,q) donde p y q son son procedimientos y x es una variable global para p y q.
- Interacción por referencia: es una interacción potencial donde los procedimientos p y q utilizan la variable x.
- Interacción por afectación: es una interacción por referencia donde el procedimiento p cambia el valor de x y el procedimiento q lo consulta. El orden de ejecución de p y q no es tenido en cuenta.
- Interacción por flujo de datos: es una interacción por afectación que permite llamar al procedimiento p a partir del procedimiento q. Este llamado es controlado por una condición sobre la variable x.

Los autores utilizan un conjunto integrado de cinco herramientas para convertir el código fuente en una descripción jerárquica del sistema [Selby88b, Hutchens87]. La primera herramienta es la encargada de extraer de cada módulo la relación de utilización entre las variables y los procedimientos. La segunda crea, a partir de los resultados de la primera herramienta, una relación de utilización única para todo el software. La tercera herramienta genera una matriz, donde cada procedimiento tiene asociado una fila y una columna, y donde el contenido de la matriz es la similitud, i.e. la cantidad de interacciones por afectación entre

Código Fuente	Interacción potencial	Interacción por referencia	Interacción por afectación	Interacción por flujo de datos
INT a,b,c,d	(p4,a,p1), (p4,a,p2), (p4,a,p3)	(p1,a,p2)	(p1,a,p2)	(p1,a,p2)
PROC p1	(p1,b,p2), (p1,b,p3), (p1,b,p4)	(p2,a,p1)	(p2,b,p1)	(p2,b,p1)
/* uses a,b */	(p2,b,p1), (p2,b,p3), (p2,b,p4)	(p1,b,p2)	(p3,c,p4)	(p3,c,p4)
/* assigns a */	(p3,b,p1), (p3,b,p2), (p3,b,p4)	(p2,b,p1)	(p4,d,p3)	(p2,e,p3)
. . . . .	(p4,b,p1), (p4,b,p2), (p4,b,p3)	(p3,c,p4)	(p2,e,p3)	
CALL p2	(p1,c,p2), (p1,c,p3), (p1,c,p4)	(p4,c,p3)		
. . . . .	(p2,c,p1), (p2,c,p3), (p2,c,p4)	(p3,d,p4)		
PROC p2	(p3,c,p1), (p3,c,p2), (p3,c,p4)	(p4,d,p3)		
/* uses a,b */	(p4,c,p1), (p4,c,p2), (p4,c,p3)	(p2,e,p3)		
/* assigns b */	(p1,d,p2), (p1,d,p3), (p1,d,p4)			
. . . . .	(p2,d,p1), (p2,d,p3), (p2,d,p4)			
CALL p3(x)	(p3,d,p1), (p3,d,p2), (p3,d,p4)			
. . . . .	(p4,d,p1), (p4,d,p2), (p4,d,p3)			
CALL p4	(p1,e,p3), (p2,e,p3), (p4,e,p3)			
. . . . .				
PROC p3 (int e)				
/* uses c,d,e */				
/* assigns c */				
. . . . .				
PROC p4				
/* uses c,d */				
/* assigns d */				
. . . . .				
START p1				

Tabla 1 - Ejemplo de interacción de datos [Hutchens85].

Matríz de interacción de datos por afectación (denotado Inter)

	p1	p2	p3	p4
p1	0	2	0	0
p2	2	0	1	0
p3	0	1	0	2
p4	0	0	2	0

Matriz de similitud (denotado Simil)

	p1	p2	p3	p4	
p1	0	1/2	I	I	I = ∞
p2	1/2	0	5/3	I	
p3	I	5/3	0	1/2	
p4	I	I	1/2	0	

Utilizando el ejemplo mostrado en la table 1, la matríz de similitud se construyó con los elementos definidos como sigue:

Simil(i,j) = (k/(n-1)) / Inter(i,j), où: n = cantidad de procedimientos

$$k = \sum \text{Inter}(i, m) + \sum \text{Inter}(m, j) - \text{Inter}(i, j),$$

m=1,..., n.

A partir de estas matrices, la técnica utlizada construye una jerarquía como sigue:

1 (p1 p 2 p3 p4)
1/2 (p1, p2)
1/2 (p3, p4)

Los grupos (p1, p2) y (p3, p4) sont logrados en la primera iteración del algoritmo. Después en cada iteración la matríz de similitud es actualizada y se forman de nuevo los grupos. Esta técnica ha sido aplicado para descomponer grandes sistemas en subsistemas. [Belady82].

Tabla 2 -Ejemplo de clasificación para las medidas de flujos de datos. [Hutchens85].

CMDDATA	0												
CMDS	27	0											
COMPILE	17	8	0										
CONCUR.	2	13	3	0									
DBLOGIC	31	23	31	4	0								
DBMNGT	42	51	10	1	25	0							
RESERV.	18	9	0	0	1	6	0						
UPDATE	5	21	52	3	30	46	2	0					
UTILITY	27	21	28	2	0	6	0	31	0				
MSG	1	0	0	0	0	0	0	0	0	0			
PROFILE	0	2	1	3	0	2	0	0	3	0	0		
SYNCH	2	5	0	0	0	4	1	0	0	0	0	0	
UTIL	8	17	9	0	5	19	4	4	14	3	1	6	0
	CMDATA	CMDS	COMPILE	CONCUR	DBLOGIC	DBMNGT	RESERV.	UPDATE	UTILITY	MSG	PROFILE	SYNCH	UTIL

Figura 1 - Matríz de similitud del sitema Smile [Maarek88a].

una pareja de procedimientos. La cuarta heramienta construye la matriz de similitud (una matriz no negativa, real, simétrica con ceros en la diagonal). La quinta herramienta utiliza una técnica estática de clasificación jerárquica [Everitt80], produce una descripción arborescente, donde los procedimientos son colocados como hojas del árbol y cada subárbol indica una clase de procedimientos que tienen en común un gran nivel de acoplamiento/cohesión. El acoplamiento es la cantidad de conexiones entre los elementos de un grupo con los otros grupos. La tabla 2 muestra un ejemplo de utilización.

3.2 Integración de Software

[Maarek88a y 88b] desarrollaron una técnica de clasificación numérica para ayudar en la integración de software de gran tamaño. Utilizaron como criterio de agrupación la cantidad de conexiones entre una pareja de modulos, es decir la cantidad de términos que son comunes a dos módulos. Para un módulo modificado, el algoritmo utilizado construye una clasificación jerárquica donde las hojas son los módulos que pueden ser afectados por la modificación y los nodos agrupan los módulos que tienen un factor de conexión más elevado.

Los autores aplicaron su técnica al sistema Smile [Kaiser87]. A partir de dos módulos de este sistema (llamados CMDS y CMDDATA, fig 1 y 2) vastamente modificados, se construyó una matriz de similitud utilizando el código fuente de Smile. Las filas y las columnas representan los módulos del sistema Smile afectados por las modificaciones en CMDS y CMDDATA. El contenido de la matriz representa el número de objetos importados entre las parejas de módulos. Por ejemplo, entre los módulos UTIL y CMDS existen 17 términos intercambiados como lo muestra la figura 1. Un algoritmo de clasificación de aridad controlada [Maarek88a] utiliza esta matriz y produce un árbol de integración. La figura 2 [Maarek88a] muestra el árbol construido a partir de la aplicación de esta técnica.

El algoritmo utilizado forma grupos de manera ascendente, el primer nivel creado esta constituido por las hojas del árbol y cada una representa un módulo. Cada interacción produce un nuevo nivel, resultado de la combinación de una pareja de módulos, de nivel inferior más aquellos que presentan una fuerte similitud (es decir que tienen el mayor número de términos intercambiados). El proceso se repite hasta obtener el árbol. Para eliminar los

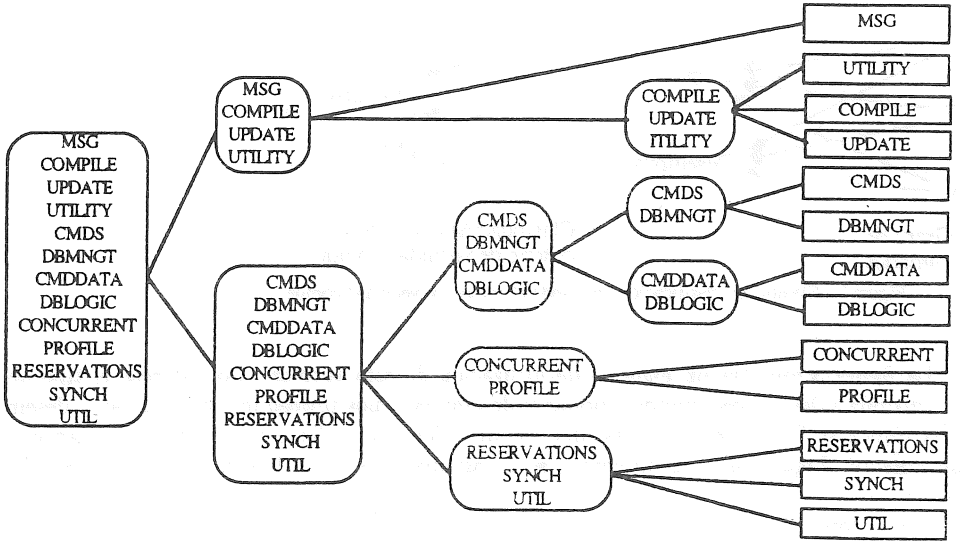


Figura 2 - Arbol de integración generado por el algoritmo de clasificación de aridad controlada.

grupos poco significativos, la dispersión estática de aridad de nuevos grupos es calculada en cada iteración por una función de varianza a partir de un valor ideal, parámetro suministradc por el usuario. La función de varianza (σ) esta definida por:

$$f_\sigma = \frac{\sum_{i=1}^{k}(x_i - \sigma)^2}{k}$$

donde:

k = número de grupos de un nivel
σ = valor ideal suministrado por el usuario
x_i = número de descendentes del grupo i en el último nivel.

En cada iteración, si el valor de σ del nuevo nivel producido es mayor que el valor del nivel inferior, el nuevo nivel queda terminado. Al final del proceso el árbol producido será una colección de niveles terminados. Por definición, el primer nivel terminado constituye las hojas del árbol.

3.3 Restructuración de sistemas viejos

[Schwanke89a y 89b] utiliza una técnica de clasificación conceptual para extraer una estructura jerárquica del software y para compararla con la estructura proyectada. El criterio de clasificación utilizado se denomina criterio de "vecindad común". A partir de un grafo de llamados a procedimientos donde los nodos son los componentes del software y los arcos las referencias cruzadas. La técnica empleada reagrupa los nodos que tienen atributos comunes.

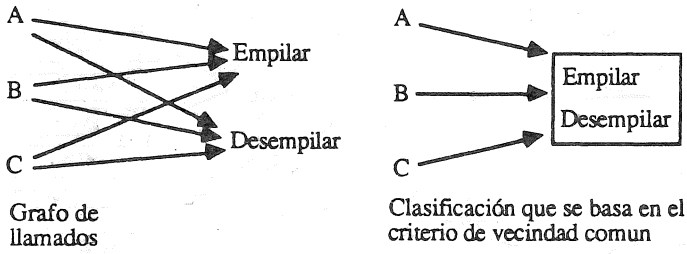


Figura 3 - Ejemplo de clasificación utilizando el criterio de vecindad común

La figura 3 muestra un ejemplo de utilización de esta técnica. Considerese el tipo abstracto pila y los procedimietos empilar y desempilar. Los procedimietos empilar y desempilar nunca se llaman entre si, asi que los módulos que utilizan una pila los llaman conjuntamente. Estos dos procedimientos serán agrupados si se utiliza el criterio de vecindad común.

Los autores utilizaron una técnica ascendente de clasificación jerárquica para poner en práctica este criterio. Esta técnica produce una descripción jerárquica del software en dos pasadas. La primera pasada construye pequeños grupos de alto grado de similitud, después los combina en grupos más grandes hasta formar un árbol de clasificación, donde las hojas son los componentes básicos del software y los subárboles los grupos. La segunda pasada elimina los subárboles de débil utilidad con el propósito de aumentar la aridad promedio y de disminur, por consequencia, la profundidad del árbol creado. La utilidad de un subárbol se mide con una función de utilidad de grupo (denotada FUG). La FUG es el producto del tamaño y de la "*pureza*" del grupo. La pureza es la probabilidad de asociación de un atributo a un miembro del grupo. Esta función favorece los grupos en los cuales la mayoríia de los miembros tienen muchos atributos en común.

La FUG utilizada por los autores es una adaptación del procedimiento de clasificación conceptual usado por el sistema Cobweb [Fischer87]. El sistema Cobweb produce un árbol de probabilidad a partir de una colección de objetos y donde la FUG es maximizada en cada nivel.

Los autores experimentaron su método sobre un software (representado por un grafo de 82 nodos y 155 arcos) descompuesto en 29 grupos. El tiempo de ejecución del algoritmo en una estación SUN de 12MB fue de 323 segundos.

3.4 Clasificación de software reutilizable

[Arango88a y 88b] utilizó el sistema Cobweb [Fischer87] para caracterizar las clases de software reutilizables. Se construyó un árbol probabilístico a partir de un conjunto de atributos que describen el software potencialmente reutilizable. Cada nodo de este árbol representa una clase de objetos que tienen características comunes. Estas características son formuladas por consultas. Una probabilidad es calculada para cada clase de objetos (representados por un nodo) teniendo en cuenta cada una de las características.

[Arango88b] aplicó la técnica sobre aplicaciones sobre un ambiente Unix. Las aplicaciones seleccionadas fueron descritas por 26 atributos. Se construyó una matriz de servicio donde las filas representaban las características resultado y las columnas el nombre de la aplicación.

	Q-68	Q-69	Q-71	Q-83	....	Q-111
Maintaining-Record-Command-History	y	y	y	y	...	y
Renaming	y	y	y	y	...	y
Saving-Command-History-at-Logout	n	n	n	n	...	n
Executing-Commands-From-File	n	n	y	y	...	n
Acces-to-Bulletin-Boards	n	n	n	n	...	n
Checking-on-Mail	n	n	n	n	...	n
Enabling-Notify-End-Of-Jobs	n	n	n	n	...	n
. . . . .	...	...	...	...	...	...
Changing-Automatic-Timing-Command	n	n	n	n	...	n

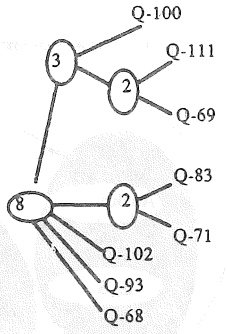


Figura 4 - Ejemplo de clasificación de aplicación utilizando el sistema Cobweb.

La figura 4 muestra una parte de la matriz construida. Por ejemplo, la primera columna, denotada Q-068, identifica el script .cshrc. Las funcionalidades de este script descritas en la matriz son: el mantenimiento de los registros de interacción del usuario con el sistema, los servicios de cambio de nombre... La matriz y la FUG del sistema cobweb fueron utilizadas para crear el árbol de clasificación de software reutilizable. Cada nodo del árbol representa un grupo de la aplicación que tienen caracteristicas parecidas.

3.5 Administración de bibliotecas de software reutilizable

[Maarek88b y 89b] desarrolló la herramienta GURU para construir automáticamente bibliotecas de software reutilizable. La herramienta permite:

- extraer los atributos significativos de la documentación del software utilizando una técnica de clasificación numérica;
- estructurar jerárquicamente las bibliotecas de código reutilizables utilizando una técnica de clasificación conceptual.

El proceso de extracción automática de atributos se hace en dos pasadas. En el transcurso de la primera pasada, los textos son analizados para encontrar las afinidades léxicas (denotadas AL) entre parejas de palabras. En la segunda pasada, los ALs significativos son seleccionados utilizando una técnica de clasificación numérica. Este algoritmo es análogo al utilizado por el módulo de particionamiento Cluster/2 1 [Michalski83]. El objetivo por alcanzar consiste en generar una cantida optimal de k-particiones sobre un conjunto de objetos, donde k es un parámetro establecido por el usuario. Este enfoque intenta reducir la cantidad de atributos producidos en la primera pasada y de identificar las clases de ALs más significativas.

La creación de bibliotecas se hace con un algoritmo de clasificación conceptual [Maarek89a] derivado de Unimen2 [Lebowitz86]. El algoritmo utiliza los componentes del software y sus atributos (gracias al proceso de indentación descrito antes). Esta técnica presenta las características siguientes:

- Hace parte de las técnicas de clasificación conceptuales llamadas jerárquicas. Identifica automáticamente las clases de una jerarquía;

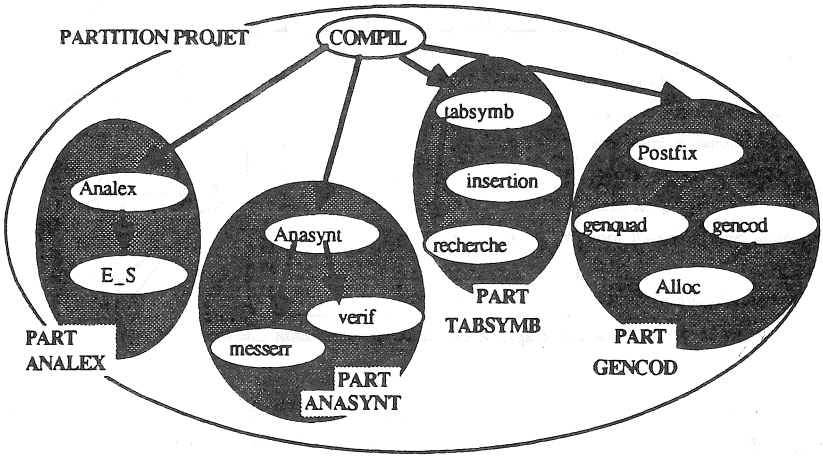


Figura 5 - Ejemplo de estructuración de software en Adèle

5 Aplicacion en el ambiente Adele

Adele es un ambiente de programación global que se centra sobre los aspectos de gestión de configuraciones, gestión de una base activa de programas y gestión de tareas.

Adele ofrece funcionalidades para soportar la evolución de los objetos, de configuraciones y versiones multiples. Estas funcionalidades son desarrolladas via un modelo de datos construido alrededor de un objeto compuesto central llamado *familia*. La noción de familia narra la evolución en versiones multiples de la noción clásica de módulo. Sin embargo, Adele ofrece una mayor libertad en la definición de la estructura intermódulos.

Diferentes mecanismos son usados por Adele para permitirle al usuario especificar las propiedades de los objetos bajo la forma de atributos y pora controlar la estructura del sistema de una manera compacta y coherente.

5.1 Estructura del software

Esta parte describe brevemente los mecanismos básicos para el modelaje de un sistema en el ambiente Adele . Para una descripción más completa ver [Belkhatir88 y 89].

El modelo Adele genera el software bajo la forma de una jerarquía de familias definiendo asi la noción de subsistema (llamadas particiones en Adele). En la figura 5 se muestra la descripción de una estructura clásica de un compilador bajo la forma de diferentes subparticiones de familias. Cada familia define separadamente sus interfaces y sus realizaciones con sus respectivas dependencias.

Los tipos predefinidos que existen en Adele ayudan a definir los objetos básicos de una familia. Estos tipos básicos tienen una semántica predefinida que permite invocar las herramientas. Estos tipos pueden ser extendidos por el usuario. La estructura del software es conservada automáticamente por las herramienta, por ejemplo, la extracción automática de dependencias. Otros tipos de elementos y de relaciones pueden ser definidas por el usuario.

5.2 Aplicación de los métodos de clasificación

La aplicación de las técnicas de clasificación en el ambiente Adele pueden verse en dos etapas de la vida del software y pueden considerarse como:

- un mecanismo de la estructura en el caso en el que las técnicas de clasificación se apliquen en la etapa de desarrollo;
- un mecanismo de verificación de la coherencia de la estructura en el caso en el que las técnicas de clasificación se apliquen en la etapa de mantenimiento.

5.2.1 Aplicación en la etapa de desarrollo

Las técnicas de clasificación ayudan a especificar la jerarquía de familias y a definir y organizar los subsistemas (particiones).

La experimentación llevada actualmente es desarrollada utilizando el método de [Schwanke89a y 89b] para producir la estructura inicial del software. Este método ha sido utilizado porque se basa en la utilización de las propiedades de los componentes del software, mecanismo que existe en Adele. Este método de clasificación produce el árbol de familias y de subsistemas potenciales.

5.2.2 Aplicación en la etapa de mantenimiento

EL software de tamaño considerable, como consecuencia de su gran tiempo de vida, se desestructura a raíz de los cambios efectuados. Son necesarios mecanismos para controlar los cambios que afectan la estructura durante el mantenimiento.

La aplicación de técnicas de clasificación permiten, en la etapa de mantenimiento, producir la estructura del software después de los cambios y confrontantarla con la inicial para detectar desviaciones y sugerir nueva estructuración. En esta etapa la técnica de clasificación puede verse como una herramienta de verificación de coherencia de la estructura del software que está en evolución.

6 Conclusion

El algoritmo de clasificación ha sido desarrollado y probado sobre la relación de dependencia entre módulos. La estructuración del software, con base únicamente en esta relación, es insuficiente. El trabajo actual pretende incorporar las propiedades atadas a cada uno de los módulos. Estamos interesados particularmente en las propiedades derivadas de las herramientas de medida. Este es otro aspecto importante en estudio: desarrollar herramientas de medida del software. Con este trabajo se podrá incorporar en la clasificación propiedades significativas como medidas de modularidad, de control de-fluxo [Adam89], etc.

Referencias

[Adam89] Jean-Michel Adam
Quality evaluation and user assistance in the context of ASPIS.
Institut IMAG - Laboratoire de Génie Informatique, rapport de recherche, March 1989.

[Arango88a] Guillermo F. Arango & E. Teratsuji
Notes on the Application of the COBWEB Clustering Function to the Identification of Patterns of Reuse.
Technical Report ASE-RTM-87, ASE, Dept. of Information and Computer Science, University of California, Irvine, CA, July 1988.

[Arango88b] Guillermo F. Arango
Domain Engineering for Software Reuse.
Technical Report 88-27 (PhD Dissertation), Depto. of Information and Computer Science, University of California, Irvine, CA, 1988.

[Belkhatir88] Noureddine Belkhatir
Nomade: un noyau d'environnement pour la programmation globale
Institut National Polytechnique de Grenoble, Décembre 1988, Thèse du Doctorat.

[Belkhatir89] Noureddine Belkhatir et Jacky Estublier. "Un gestionnaire d'activités de programmation globale pour Nomade." In *Proc. of the Second Internacional Workshop Software Engineering & Its Applications.* Toulouse (France), Dec. 4-8, 1989. p. 703-716.

[Everitt80] B. S. Everitt
Cluster Analysis, 2nd ed. Heineman Educational Books, London, 1980.

[Fischer86] Douglas H. Fischer & Pat Langley
"Methods of conceptual clustering and their relation to numerical taxonomy."
In *Artificial Intelligence and Statistics*, W. Gale (Ed.), Reading, Addison-Wesley, 1986.

[Fischer87] Douglas H. Fischer
"Knowledge acquisition via incremental conceptual clustering."
Machine Learning, 2(2):139-72, 1987.

[Hutchens85] David H. Hutchens & Victor R. Basili
"System structure analysis: clustering with data bindings."
IEEE Transactions on Software engineering, SE-11(8):749-57, Aug. 1985.

[Hutchens87] David H. Hutchens
Software Tools for Data Bindings Analysis.
Technical Report, Depto. of Computer Science, Clemson University, Clemson, SC, 1987.

[Kaiser87] Gail E. Kaiser & Dewayne Perry
"Workspaces and experimental databases: automated suport for software maintenance and evolution."
In *Proc. Conference on Software Maintenance*, Austin, September 1987, p. 108-14

[Lebowitz86] M. Lebowitz
"Concept learning in a rich input domain: generalization-based memory."
In *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, vol. 2, J.G. Carbonell e T.M. Mitchell (Eds.), Morgan Kaufman, Los Altos, CA, 1986. pp. 193-214.

[Maarek87] Yoelle S. Maarek & Gail E. Kaiser
"Using conceptual clustering for classifying reusable ada code."
In *Proc. of the ACM SIGAda International Conference*, Boston, MA, Dec. 1987. Issue of the Ada Letters, ACM Press. p. 208-15.

[Maarek88a] Yoelle S. Maarek & Gail E. Kaiser
"Change management for very large software systems."
In *Proc. of the Seventh Annual International Phoenix Conference on Computers and Communications*, Phoenix, AZ, USA, March 16-18 1988. p. 280-5. IEEE Computer Society Press, Washington, DC.

[Maarek88b] Yoelle S. Maarek
"On the use of cluster analysis for assisting maintenance of large software systems."
In *Proc. of the Third Israel Conference on Computer Systems and Software Engineering*, Tel Aviv, Israel, June 6-7 1988. p. 178-86. IEEE Computer Society Press, Washington, DC.

[Maarek89a] Yoelle S. Maarek
Concept learning via conceptual clustering.
Research Report 14743, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY. June 1989.

[Maarek89b] Yoelle S. Maarek; Daniel M. Berry; Gail E. Kaiser
Automatically generating software libraries without pre-encoded knowledge.
Research Report 14990, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY. Aug. 1989.

[Michalski80] Ryszard S. Michalski
"Knowledge acquisition through conceptual clustering: a theoretical framework and algorithm for partitioning data into conjunctive concepts."
In *International Journal of Policy Analysis and Information Systems*, 4(3):219-43, 1980.

[Michalski83] Ryszard S. Michalski & Robert E. Stepp
"Automated constructions of classifications: conceptual clustering versus numerical taxonomy."
In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(4):396-409, 1983.

[Michalski87] Ryszard S. Michalski & Robert E. Stepp
"Learning from observation: conceptual clustering."
In *Machine Learning: an artificial intelligence approach*, Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell (Eds),Tioga, 1987. pp. 331-63.

[Newbery89] Frances Newbery
"Edge concentration: a method for clustering directed graphs."
In *Proc. of the Second Internacional Worksohop on Software Version and Configuration Control*. Princeton, Oct. 24-27, 1989. In *ACM SIGSOFT Software Engineering Notes*, 17(7):76-85, Nov. 1989.

[Pressman87] Roger S. Pressman
Software Engineering: a practitioner's approach. New York, McGraw-Hill, 1987.

[Schwanke89a] Robert W. Schwanke; R. Altucher; Michael A. Platoff
"Discovering, visualizing, and controlling software structure."
ACM SIGSOFT Engineering Notes, 14(3):147-50, Pittsburg, May 1989.

[Schwanke89b] Robert W. Schwanke & Michael A. Platoff
"Cross references are features."
In *Proc. of the Second International Workshop on Software Version and Configuration Control.* Princeton, Oct. 24-27, 1989. In *ACM SIGSOFT Software Engineering Notes*, 17(7):86-95, Nov. 1989.

[Selby88a] Richard W. Selby & Victor R. Basili
Analyzing error-prone system coupling and cohesion.
Technical Report, Dept. of Information and Computer Science, University of California, Irvine, CA, 1988.

[Selby88b] Richard W. Selby & Victor R. Basili
"Error localization during software maintenance: generating hierarchical system descriptions from the source code alone."
In *Proc. of the International Conference on Software Maintenance*, Phoenix, October 24-27, 1988, p. 192-7.

[Sokal63] Robert R. Sokal & P. H. A. Sneath
Numerical Taxonomy. Z. H. Freeman, San Francisco, 1963.

[Sokal88] Robert R. Sokal
"Unsolved problems in numerical taxonomy."
In *Classification and Related Methods of Data Analysis*, H. H. Bock (Ed.). Elsevier, North-Holland, 1988.

XVII LATIN AMERICAN
INFORMATICS CONFERENCE
PANEL 1991
Agosto 8-12, 1991

La Evaluación de Desempeño en una Metodología de Desarrollo de Sistemas de Información Distribuidos

Jose AGUILAR (1) - **Alberto VALDERRUTEN** (2)

(1) Grupo MaMuD - Departamento de Computación - Facultad de Ingeniería
Universidad de los Andes - MERIDA - VENEZUELA

(2) Institut de Recherche en Informatique de Toulouse
Equipe SIERA - Université Paul Sabatier
118, Route de Narbonne 31062 TOULOUSE Cédex - FRANCE

teléfono: (+33) 61.55.67.68
FAX: (+33) 61.52.14.58
e-mail: aguilar@irit.fr - valderru@irit.fr

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 más 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.

I. LA EVALUACION DE DESEMPEÑO

La evaluación de desempeño es la mejor manera para analizar el comportamiento de un sistema. Puede ser visto como [VAL 89]:

- la medida de la calidad de funcionamiento del sistema, es decir, su aptitud para realizar las tareas que le son pedidas;
- la medida de la calidad de servicio, traducida como la satisfacción del usuario visto en términos de tiempo (plazos de espera, tiempos de respuesta);
- y la medida de la utilización correcta de sus recursos para una determinada carga de trabajo.

Si el sistema no existe aún, esta técnica es la única manera de evaluar su posible comportamiento, actividad justificada por razones de costo, tiempo y la imposibilidad de saber de antemano las tareas que deberá soportar. Si el sistema existe, se pueden hacer evaluaciones de condiciones críticas de funcionamiento (cuellos de botella, abrazo mortal, cargas extras de trabajo...) antes de su ocurrencia real en el sistema, pudiéndose así saber cual sería su comportamiento en esos casos y determinar los ajustes necesarios. Estas evaluaciones dan información sobre los criterios de desempeño del sistema para cargas de trabajos determinadas.

La evaluación de desempeño permite responder a preguntas que normalmente aparecen durante el desarrollo de sistemas de información distribuidos, tales como [BON 90]:

¿ Cómo mejorarlos y optimizarlos?
¿ Cómo evitar y prevenir posibles fallas?
¿ Se aprovechan todas sus posibilidades?
¿ Se pueden utilizar mejor sus recursos?

Todo esto hace necesaria la incorporación de los resultados de la evaluación de desempeño en el proceso de toma de decisiones durante el desarrollo de tales sistemas. Estos resultados permiten [FDI 89]:

- Predecir o estimar su desempeño, sobretodo si el sistema no existe.
- Tener información técnica que asista al ingeniero en la escogencia de una alternativa.
- Entender mejor el sistema en estudio, lo que reduce las diferencias entre los requerimientos iniciales y el sistema final.
- Evaluar los posibles componentes del sistema, para escoger los mas adaptados a los requerimientos que éste debe cumplir.
- Observar el comportamiento del sistema y determinar así una óptima utilización.
- Representar el sistema con hardware y/o software que aún no tiene incorporado, para determinar su impacto.

- Permitir un proceso iterativo de diseño, evaluando las restricciones de desempeño y haciendo las modificaciones del caso.

- Disminuir los riesgos en el desarrollo, descubriendo fallas que pueden ser corregidas a tiempo, lo que evita problemas futuros.

- Dimensionar correctamente el sistema.

- Validar el sistema.

- Verificar la factibilidad de realizar un sistema que responda a las especificaciones pedidas.

- Identificar los componentes más sensibles del sistema a nivel de desempeño, para preveer su vigilancia.

- Disponer de una representación muy cercana al sistema, para diversos usos.

- Predecir el comportamiento del sistema bajo cargas de trabajos especifícas.

II. METODOS DE EVALUACION DE DESEMPEÑO:

Para hacer el análisis del desempeño de un sistema hay que comprenderlo, y si es posible, medirlo. Existen varios métodos, pero sólo presentamos aquellos que usen una representación del sistema para ese estudio, por su utilidad durante el desarrollo del sistema.

Estos métodos tienen como característica principal la cuantificación del desempeño del sistema, bien sea por estimación o por observación. Estos valores (tiempos de respuesta, tasa de uso de los recursos,...) traducen los criterios de interés para su análisis.

II.1 Métodos Analíticos:

Son métodos que sustituyen el sistema real por un modelo que representa su funcionamiento de manera más o menos detallada. El sistema es transformado en un conjunto de ecuaciones matemáticas que capturan los componentes que caracterizan su comportamiento. Son métodos muy eficaces si se puede deducir con facilidad dichas ecuaciones, vistas como un conjunto de relaciones que ligan la evolución de las variables que caracterizan al sistema.

Los métodos analíticos se caracterizan por [FDI 89]:

- Requieren una gran cantidad de tiempo y esfuerzo. La deducción de cada elemento del modelo necesita un trabajo exhaustivo de análisis.

- Es necesario conocer muy bien el comportamiento del sistema. Esto dificulta la aplicación de estas técnicas para estudios de sistemas complejos.

- Si el sistema es muy complejo, hay que hacer asunciones de simplificación.

En este grupo encontramos:

II.1.1 Redes de Petri Estocásticas

Es una versión de Redes de Petri, pero introduciendo la noción de tiempo al asociar plazos a las transiciones en los modelos de los sistemas. Sirve para evaluar problemas de sincronización y paralelismo.

II.1.2 Canales de Markov

Es un método que describe la evolución del sistema a partir de sus estados. Esto hace necesario especificar la información que describe cada estado y calcular las tasas de transiciones entre ellos. Un estado depende únicamente de sus estados precedentes. Si hay un gran número de estados este método es inadaptado.

II.1.3 Redes de colas de espera

Es el método analítico mas usado en la modelización de sistemas de información distribuidos, por ser una manera muy natural de representarlos. La modelización puede ser vista como:

- Un conjunto de recursos, componentes del sistema necesarios para realizar una actividad. Estan formados por una cola de espera donde se acumulan los clientes que desean acceder a él y por un servidor que provee el servicio.
- Un conjunto de clientes, entidades que circulan sobre la red pidiendo servicios a los recursos.
- Un conjunto de actividades, traducidas por el proceso de evolución de cada cliente en el sistema.

Sólo se conocen soluciones matemáticas a ciertos modelos, (redes de Jackson, redes BCMP), lo que hace necesario que para poder ser usadas en la modelización de un sistema, haya que asumir ciertas restricciones en su comportamiento.

II.2 Simulación

Es la única vía de estudiar un sistema complejo y de aproximarse a su comportamiento transitorio. En cuanto al estudio de comportamientos estacionarios, se necesita un importante

trabajo de interpretación de sus resultados por medio de técnicas de cálculo estadístico (intervalos de confianza...).

Tiene como ventaja su flexibilidad con respecto a los anteriores, pero como desventajas su largo tiempo de ejecución y la complejidad del análisis sobre los resultados obtenidos.

III. METODOLOGIA DE DESARROLLO DE SISTEMAS DE INFORMACION DISTRIBUIDOS [BOD 89], [CAS 86], [LES 88]

Para los estudios sobre los nuevos sistemas de información, se deben aprovechar las experiencias adquiridas durante desarrollos anteriores, lo que permite concebir metodologías que permitan salir del empirismo al asegurar una definición precisa de las tareas a realizar.

La metodología que presentamos es general, haciéndose hincapié en todos los aspectos relativos a un ambiente distribuido, en el cual tomamos en cuenta conceptos de cooperación y de comunicación. Esto nos permite hacer un análisis teórico sobre la utilización de la evaluación de desempeño, que puede adaptarse al desarrollo de sistemas de información distribuidos particulares (bases de datos distribuidas, sistemas operativos distribuidos...).

Las etapas de esta metodología son:

III.1 Esquema Director

Esta etapa consiste en un análisis global de la situación existente en la organización y en su ambiente, con el fin de establecer los objetivos estratégicos, sus necesidades de información y la factibilidad de realizar el sistema. Se puede descomponer en:

- Macroanálisis, donde se realiza un análisis global de la situación en la organización y en su entorno.
- Estudio de oportunidad, donde se determina la factibilidad funcional, técnica y económica del sistema.

III.2 Análisis Conceptual:

Su objetivo es obtener un sólido conocimiento sobre la organización, para poder definir de manera precisa las necesidades que debe cubrir el sistema a construir. Puede ser descompuesta en:

- Análisis de la situación: es un estudio detallado y preciso de la organización. Se debe tomar en cuenta su implantación geográfica, la eventual existencia de una infraestructura informática, el ambiente de trabajo (puestos de trabajo, tráfico de información,...), las políticas de tratamiento y almacenamiento...

- Análisis de restricciones: se definen todas las restricciones que debe cumplir el nuevo sistema a nivel: técnico, económico, de personal, de operación, geográfico, de desempeño (tiempos de respuesta), administrativo o de factores externos a la organización.

- Análisis de necesidades: consiste en hacer una descripción de los servicios que esperan los usuarios. Esta tarea debe ser realizada en conjunto por los usuarios y los ingenieros que van a construir el sistema.

III.3 Análisis funcional

En esta etapa se empiezan a definir los requerimientos de los usuarios a partir de los resultados obtenidos en las etapas anteriores. Esto conllevara a establecer las especificaciones funcionales del nuevo sistema, es decir, sus funciones y sus atributos. Las especificaciones funcionales interpretan cualitativamente y cuantitativamente las necesidades de los usuarios mediante la descripción de los eventos a tratar, los tratamientos a cumplir sobre los eventos, la lista de datos necesarios para los tratamientos, las características del hardware y software a utilizar y los resultados a obtener.

III.4 Estudio de alternativas

En esta etapa se definen con detalles las soluciones posibles que satisfagan las especificaciones definidas en las etapas anteriores, y se decide cual se escoge. Esta etapa se descompone en:

- Definición de las alternativas, definiendo para cada una: equipos y software necesario, ambiente de implantación y resultados que se pueden obtener particularmente a nivel de desempeño.

- Selección de la mejor alternativa a partir del estudio de las restricciones y de las especificaciones funcionales.

III.5 Diseño

Su objetivo es realizar la descripción detallada de la solución escogida, en vistas de su implantación. Puede ser descompuesta en:

- Definición de la organización lógica y física de los datos, definiendo las características de los medios de almacenamiento, el modo de organización, la estructura de los registros, el acceso a los archivos, los soportes de almacenamiento, la política de administración de datos (centralizado, repartido por copias, repartido por pedazos, ...).

- Diseño de las entradas/salidas del sistema, decidiendo el tipo de diálogo hombre-máquina, la estructura de las pantallas, las modalidades de salida...

- Diseño del sistema programado, definiendo su descomposición funcional, sus características (soportar varias tareas, trabajar en ambiente repartido, ...) y sus algoritmos.

- Definición de la arquitectura del sistema a nivel organizacional: se debe definir la distribución geográfica de los sitios de trabajo, la repartición de los datos y tratamientos, la arquitectura del sistema administrativo y la arquitectura de comunicación (mecanismos de sincronización, exclusión mutua, elección,...).

III.6 Implementación

Es la etapa de construcción del sistema. Puede ser descompuesta en:

- Escoger los componentes del sistema, determinando los diferentes elementos que lo compondrán a nivel de hardware (terminales, unidades de almacenamiento y tratamiento, medios de transmisión...) y software (sistema operativo, protocolos de comunicación, manejador de las bases de datos...).

- Elaborar el sistema programado, construyendo los programas identificados para satisfacer las demandas específicas de los usuarios.

III.7 Verificación y puesta a punto

Esta etapa tiene por objetivo verificar los resultados del sistema viendo las discordancias que existan con los objetivos, restricciones y requerimientos definidos al inicio del desarrollo, poniendo especial atención a su comportamiento a nivel de comunicaciones. Se hacen las correcciones del caso para eliminar esas diferencias. Los trabajos a realizar son:

- La construcción de casos de prueba, especificando el objetivo de la prueba, las actividades a realizar sobre el sistema, los requerimientos asociados y los criterios de éxito.

- El test del sistema con los casos de prueba definidos, ejecutando las actividades ya preestablecidas en cada uno y observando su comportamiento.

- La optimización del sistema, mediante las modificaciones que se requieran según los resultados de las pruebas.

III.8 Implantación

Es la etapa donde se realiza la instalación del sistema en la organización. Comprende:

- Las preparaciones previas, donde se definen las etapas transitorias, es decir, la cohabitación entre los sistemas viejo y nuevo, las conversiones necesarias entre los dos (por ejemplo, a nivel de archivos), la presentación del sistema, la formación de los usuarios y los trabajos iniciales de carga de archivos, instalación del software,...
- El lanzamiento, donde se inicia el periodo normal de funcionamiento del sistema.

III.9 Mantenimiento:

Consiste en hacer evolucionar el sistema en función de las nuevas necesidades, de las deficiencias observadas en su operación, de los cambios en el ambiente y de los progresos tecnológicos. En esta etapa se debe hacer un seguimiento de la administración global del sistema, la cual está definida por las 6 funciones establecidas en el modelo de administración propuesto por la OSI: configuración, desempeño, seguridad, nombres y direcciones, fallas y contabilidad.

Los pasos a realizar son:

- Estudio de impacto: localizar y evaluar las adaptaciones que se deben hacer sobre el sistema
- Ejecución de las adaptaciones.
- Test sobre el sistema modificado para verificar si la acción de mantenimiento no produjo perturbaciones indeseables sobre el resto del sistema.

IV. CRITERIOS DE DESEMPEÑO CARACTERISTICOS DE LOS SISTEMAS DE INFORMACION DISTRIBUIDOS

Pensar en medidas generales de desempeño es bastante difícil, ya que la práctica de tener en cuenta medidas de desempeño para administrar un sistema es reciente. Presentamos aquí los principales criterios de desempeño normalmente aceptados. Podrán encontrarse otros criterios particulares a un sistema distribuido bajo estudio.

Los criterios de desempeño son descriptores utilizados para representar las diferentes características del sistema a nivel de desempeño, lo que les da cierta subjetividad al poderse establecer diferentes conjuntos de criterios de acuerdo a un punto de vista. Pero esta noción puede ganar en objetividad al definirse las caracterísiticas generales a observar.

De estos criterios, existen algunos perceptibles por los usuarios y otros internos al sistema, algunos tienen que ver con la productividad y otros con su utilización. No pretendemos aquí hacer una clasificación de estos, mas bien una enumeración: [FER 78],[PUJ 89],[BUI 78]

- *Throughput*: medida de la cantidad de trabajo útil realizado por el sistema por unidad de tiempo

- *Capacidad*: volumen máximo de información tratada por unidad de tiempo bajo una carga de trabajo balanceada (situación en la que un conjunto de tareas activan todos los recursos del sistema).

- *Tiempo de respuesta*: tiempo que transcurre entre el momento en que el usuario hace un pedido al sistema y la aparición de la respuesta respectiva.

- *Taza de utilización de un elemento del sistema*: relación entre el tiempo de utilización dentro de un intervalo de tiempo y la duración de dicho intervalo.

- *Número de usuarios simultáneos* que pueden haber en el sistema en un momento dado.

- *Tiempo de respuesta de un recurso del sistema*: está compuesto por el tiempo de espera para su acceso y el tiempo de ejecución de la tarea pedida.

- *Plazos para el encaminamiento de mensajes*: está compuesto por los tiempos de espera en los nodos de comunicación y los tiempos de transmisión.

- *Tiempo de acceso al medio de comunicación*.

- *Estabilidad*: tiempo promedio que transcurre antes que se pueda transmitir con éxito un paquete de información.

- *Flexibilidad*: capacidad de extensión y de adaptación a nuevas tareas.

- *Caudal de los canales de comunicación*: tráfico efectivo permitido sobre éstos.

- *Overhead*: cantidad de tiempo de procesador y de canales pedido por el sistema operativo frente a la cantidad de recursos pedidos por los usuarios.

- *Satisfacción de los usuarios*

- *Confiabilidad del sistema*: número de fallas y errores encontrados durante su operación.

- *Facilidad de utilización del sistema*: tiempo necesario para preparar una tarea sobre el sistema o tiempo necesario para utilizarlo. Se puede incluir también el tiempo de aprendizaje.

- *Disponibilidad*: tiempo promedio durante el cual el sistema puede ser utilizado sin problema.

- *Calidad de funcionamiento*: capacidad del sistema para ejecutar de una manera correcta las tareas que le son pedidas.

Existen criterios más importantes que otros en un estudio en particular. Por ejemplo, el tiempo de respuesta para aplicaciones interactivas y el caudal de los canales de comunicación para aplicaciones con mucha información a transportar.

V. LA EVALUACION DE DESEMPEÑO EN LAS ETAPAS DE DESARROLLO DE UN SISTEMA DE INFORMACION DISTRIBUIDO

En esta parte presentamos un estudio del uso de la evaluación del desempeño en las etapas de la metodología propuesta, para una mejor ubicación de las relaciones entre los dos. Las proposiciones que se presentan serán aplicadas sobre un tipo de sistemas de información distribuidos tomando en cuenta sus necesidades particulares. Bajo este contexto de trabajo se llega a proponer una herramienta de ayuda al desarrollo que toma en cuenta las características de desempeño de este tipo de sistemas.

V.1 Esquema director:

Al usar algún método de evaluación de desempeño para el análisis global de la organización se permite formalizar una representación que conlleva a una mayor comprensión de su comportamiento para determinar sus necesidades a nivel de información, los problemas presentes en ella y las restricciones ligadas a su ambiente.

Se permite de este modo determinar la factibilidad de realizar un sistema de información y si es el caso, sus objetivos.

V.2 Análisis conceptual

En esta etapa se continua con el análisis de la organización para llegar a un sólido conocimiento de ella al detallar su representación. Durante este estudio se debe poder definir en la representación:

- la estructura informática de la organización (repartición geográfica de los medios de almacenamiento y tratamiento, medios de comunicación,...),
- la descripción de su ambiente de trabajo (puestos de trabajo, intercambios conversacionales, datos en los intercambios, ...).

La representación nos permite definir las restricciones de desempeño de la organización de acuerdo a sus necesidades y las políticas de administración y mantenimiento apropiadas.

Hasta aquí llega el estudio para conocer la organización de manera a encontrar sus problemas, sus necesidades y en general la factibilidad de desarrollar un sistema de información como medio de solución. Este estudio se usa para definir los objetivos del nuevo sistema.

V.3 Análisis funcional

Se comienza aquí con el estudio de las especificaciones funcionales del nuevo sistema. La evaluación de desempeño es un medio de representación de las especificaciones que permite comprenderlas bien desde un punto de vista temporal, además de determinar si pueden ser satisfechas por el nuevo sistema o si estan bien fundamentadas.

Estas representaciones seran comparadas con el sistema finalmente implementado. Si se logra construir una herramienta que realice esta tarea, se podrá llegar a validar automáticamente la calidad de funcionamiento del sistema.

V.4 Estudio de alternativas:

La evaluación de desempeño es el mejor medio para representar las posibles soluciones definidas en esta etapa para comprenderlas bien, puesto que es una herramienta segura para hacer el análisis sobre la calidad de desempeño de cada alternativa, dando información técnica muy precisa que permita la selección de la mejor solución, de manera a evitar problemas futuros a nivel de costos, retardos en el desarrollo, subdimensionamiento o sobredimensionamiento del sistema.

V.5 Diseño

La representación de la solución escogida en la etapa anterior debe ser detallada a nivel de:

- la organización de los datos y tratamientos (repartición, caracterísiticas que deben tener los soportes,...),
- las políticas de operación (multiprogramación, sistemas centralizados o repartidos de almacenamientos, ...),
- la distribución de los puestos de trabajo,
- la topología de la red,
- la definición del sistema de administración.

Todo esto conforma la arquitectura del sistema a construir.

Con la descripción obtenida se estima el desempeño de esta representación para asegurar que satisfaga las restricciones. Seguidamente se hacen las modificaciones del caso sobre la representación para mejorar su diseño. De esta manera se realiza un proceso de diseño iterativo, partiendo de una solución inicial que se va mejorando para ir produciendo

aproximaciones cada vez mas cercanas a las especificaciones pedidas. Al final de este proceso se obtiene una descripción que se puede implementar inmediatamente.

Es necesario parametrizar en la representación del sistema sus características mas resaltantes desde un punto de vista cuantitativo (por ejemplo, para los canales de comunicación, el caudal permitido), para poder evaluar las diferentes alternativas.

Esta representación del sistema es un medio para evaluar las respuestas a las preguntas que surgen durante la concepción de un sistema. Es la única manera de evaluar un sistema en curso de realización.

V.6 Realización

Esta etapa también puede aprovechar los resultados de la evaluación de desempeño al momento de escoger los componentes del nuevo sistema, por ser un proceso de selección donde se debe utilizar la representación hecha anteriormente, dándole valores a los parámetros según las características de los componentes candidatos.

Es un medio seguro y efectivo de evaluación para la selección. Sus resultados permiten proponer modificaciones, ya sea por la necesidad de adaptar el sistema a las alternativas existentes en el mercado, o por la posibilidad de mejorar el diseño original.

V.7 Verificación

La primera verificación a realizar es comprobar que las especificaciones funcionales son cumplidas por la representación que se viene de hacer. Es un proceso automatizable, traduciendo las especificaciones antes definidas en la carga de trabajo que debe aplicarse a la representación. Si no es aplicable o se observa una degradación del sistema, se concluye que las especificaciones no se cumplen.

Una segunda verificación permite evaluar el funcionamiento del sistema con el fin de optimizarlo, suponiendo hipotéticas cargas de trabajo para encontrar problemas inesperados no detectados. La idea es estudiar el comportamiento del sistema bajo cargas de trabajo excepcionales.

Estas verificaciones no pueden ser puntuales: es indispensable ver la influencia de los parámetros sobre los costos y el desempeño del sistema. Por ejemplo, para determinar el caudal de un canal, tenemos que ver la evolución de los costos en función de los tiempos de respuesta, para justificar no solo la arquitectura del sistema, sino también su costo.

V.8 Operación

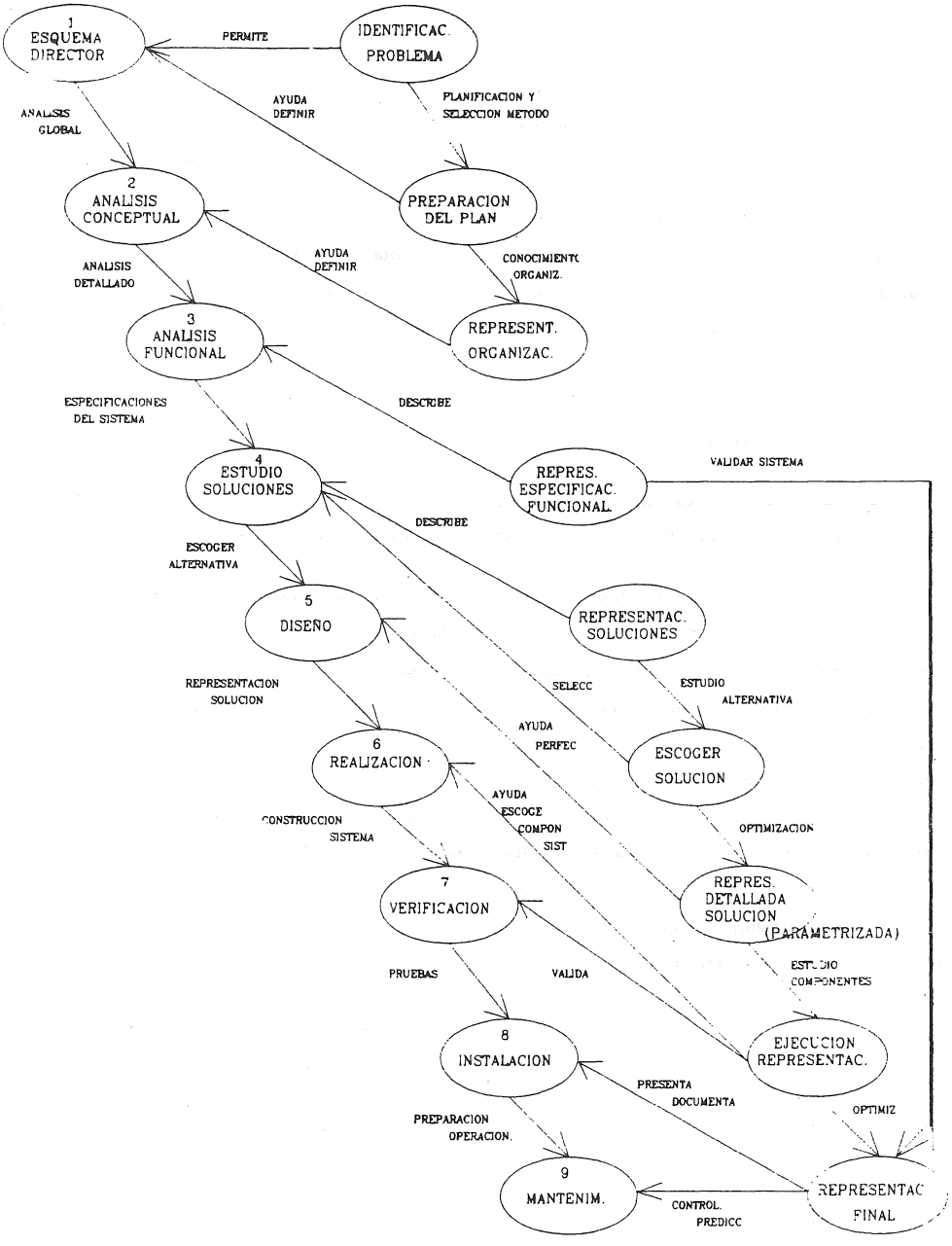
Con las representaciones hechas en las dos primeras y últimas etapas, se definen las conversiones que se deben realizar ya sea a nivel de equipos, actividades o del sistema programado. Este proceso de integración de representaciones puede ser automatizado.

Por otra parte, la última representación es la mejor documentación que se puede tener sobre el sistema construido, permitiendolo presentar de una manera comprensible a los futuros usuarios (una demostración del futuro sistema a través de su representación). Con ella se muestra tanto su arquitectura como su comportamiento funcional.

V.9 Mantenimiento

La evaluación de desempeño juega aquí un papel de vigilancia y de predicción del funcionamiento del sistema. Es un ambiente ideal para los estudios de su optimización.

Antes de hacer alguna modificación en el sistema real, se realiza de antemano el estudio sobre su representación. Por ejemplo, para optimizarlo, analizamos en la representación los cambios a efectuar y estudiamos el impacto causado por estos. Si el estudio es de predicción, asumimos hipotésis de trabajo sobre el sistema (como una carga de trabajo que obligue a usar todos sus recursos) y las aplicamos en la representación para observar su comportamiento y tomar las decisiones que permitan prevenir los problemas encontrados. Esta es una herramienta que puede ser utilizada en la administración de los recursos del sistema, en la simulación de escenarios de fallas, en la modificación de su configuración... de una manera flexible sin que haya una repercusión directa sobre el sistema real. Así visualizamos las tareas a realizar para la evolución del sistema, adaptándolo a las necesidades de su ambiente. Es una herramienta poderosa para el seguimiento y la administración del sistema.



RELACION ENTRE LA METODOLOGIA DE DESARROLLO DE SISTEMAS DE INFORMACION DISTRIBUIDOS Y LA EVALUACION DE DESEMPEÑO

VI. ESPECIFICACION DE UNA HERRAMIENTA DE AYUDA AL DESARROLLO DE SISTEMAS DE INFORMACION DISTRIBUIDOS [CON 89], [QNA 90]

Este estudio metodológico nos permite proponer la especificación de herramientas que ayuden al ingeniero en las tareas de desarrollo de sistemas de información distribuidos. Es así como nuestro equipo IRIT/SIERA lleva a cabo investigaciones en este campo, que comprenden el proyecto ESPRIT COMPLEMENT para sistemas de tiempo real y el proyecto MAITE para sistemas de comunicación.

En este apartado, presentamos las especificaciones de MAITE, una herramienta de evaluación de desempeño útil para el desarrollo de una clase de sistemas de información distribuidos: los sistemas de comunicación. Este proyecto caracteriza el esfuerzo entre la industria y los laboratorios de investigación en esta área. La compañía MATRA-Espace realizó el estudio de necesidades y el laboratorio IRIT/SIERA ofreció el soporte teórico y los resultados de sus investigaciones en el campo de los sistemas de comunicación.

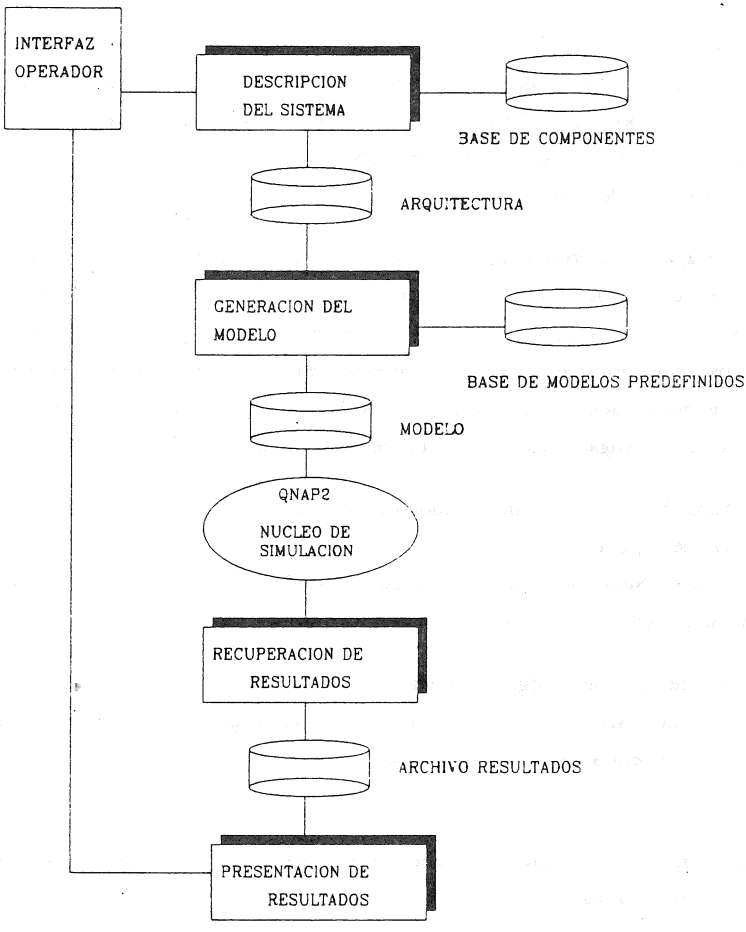
Para MAITE se escogió como método de evaluación de desempeño, la modelización por redes de colas de espera y como apoyo a este método se utilizó el lenguaje de modelización QNAP2 (Queieng Network Analysis Package). QNAP2 está compuesto de un lenguaje de descripción de modelos y de un conjunto de métodos de resolución, entre ellos la simulación.

Después de haberse escogido el método de evaluación, la herramienta asociada y las necesidades de tales sistemas, se pasa a definir el soporte programado de ayuda al ingeniero, de modo que los problemas de modelización sean transparentes, a través de una interfaz que permita:

- definir rápida y fácilmente un modelo del sistema a nivel de hardware y software,
- generar automáticamente los modelos en QNAP2,
- activar y controlar la simulación,
- recuperar los resultados de la simulación en vista de su análisis.

La parte más compleja de MAITE es la generación automática de los modelos. La solución consistió en crear una base de modelos genéricos y parametrizables, agrupados en familias (procesadores, unidades de almacenamiento, protocolos de comunicación...) de modo a adaptarse a los diferentes casos de descripción. Cada uno de estos macro-modelos es una sub-red de colas de espera, y para construir el modelo global es necesario realizar la conexión de varios de ellos.

MAITE puede ser visto como un sistema programado en torno al núcleo QNAP2 que ofrece a los usuarios toda la potencia de la modelización y simulación a través de un interfaz gráfica convivial.



ARQUITECTURA FUNCIONAL DE MAITE

Una sesión de trabajo con MAITE consiste en:

- *Descripción del sistema* usando los componentes de hardware que tiene la base de modelos, se fijan los valores de sus parámetros y se van colocando en el esquema que se va visualizando en la pantalla. Estos componentes son después interconectados para construir el

modelo físico general. Las actividades de los procesos programados son descritas gracias a funcionalidades que pueden especificarse en los componentes.

- *Generación del modelo*, interpretando la descripción anterior para generar un modelo en el lenguaje QNAP2. El modelo es construido usando las estructuras predefinidas y parametrizadas guardadas en la base.

- *Simulación del modelo*, gracias a las funciones del núcleo de QNAP2. Los parámetros de la simulación (duración, unidades a observar...) son definidos interactivamente por el usuario.

- *Recuperación de los resultados*, la cual se puede realizar de dos maneras: en lote, en la que no hay comunicación directa entre QNAP2 y MAITE sino que los resultados, guardados en un archivo, son recuperados posteriormente para su presentación, o interactivamente, en la que el usuario controla el proceso a través de la interfaz y los resultados son visualizados en tiempo real.

- *Presentación de los resultados*, permitiendo la interpretación de los datos gracias a las funcionalidades que los transforma en gráficas, tablas y curvas.

VI. CONCLUSION

Este artículo muestra el aporte de la evaluación de desempeño para determinar la arquitectura de un sistema de información distribuido, cuyos aspectos más relevantes a tomar en cuenta son su costo y comportamiento. Esta actividad abre nuevas perspectivas de investigación, que parte de la toma de medidas sobre los sistemas para estudiar su comportamiento, con el fin de incluir estos resultados en las metodologías de desarrollo.

La metodología propuesta para la construcción de sistemas de información distribuidos es una guía, la cual se presenta como un proceso iterativo y progresivo que va de un nivel abstracto hasta llegar a un nivel de detalle bien explícito. Por situarnos en un ambiente distribuido, es necesario utilizar técnicas de evaluación de desempeño, que pueden ser usadas en las diversas etapas del ciclo de vida del sistema como soporte de control y estudio.

Nuestro convencimiento de la necesidad de utilizar la evaluación de desempeño durante el desarrollo de un sistema de informacion distribuido, nos obligó a estudiar los métodos de evaluación existentes. Partimos de los métodos que usan una representación del sistema para realizar este estudio, por la posibilidad de reutilizarla y mejorarla conforme vayamos avanzando en su desarrollo. Para poder usarlo en la metodología, el método debe presentar una flexibilidad y facilidad de utilización, necesitar poco conocimiento a priori para ser usado y permitir la definición de modelos complejos.

Logrando automatizar la evaluación de desempeño dentro de esta metodología, se aprovechan sus resultados en los procesos de selección, optimización, planificación y

presentación implícitos en el desarrollo. MAITE muestra la factibilidad de construir una herramienta de evaluación de desempeño que responda a las exigencias de los diseñadores de sistemas de información distribuidos, en este caso particular, los sistemas de comunicación.

En investigaciones futuras, las proposiciones generales hechas a nivel de la utilización de la evaluación de desempeño en las etapas de la metodologia propuesta, son un punto de partida para estudios especifícos de otros sistemas de información distribuidos particulares (bases de datos distribuidas, sistemas operativos repartidos...).

VII. BIBLIOGRAFIA

[BOD 89] F. Bodart, Y. Pigneur, **Conception assitée des systèmes d'information: méthode - modèles, outils**, Masson, 1989.

[BON 90] F. Bontemps, **L'analyse de performances pour l'implantation d'un systeme de communication**, document interne CERFIA/SIERA, 1990.

[BUI 78] C. Bui, P. Bourret, **Mesure des systemes informatiques**, Cepadues Editions, 1978.

[CAS 86] X. Castellani, **Méthode générale d'analyse d'une application informatique. Tome I, II et III**, Masson, 1986.

[CON 89] E. Conquet, **MAITE: Un outil d'aide à l'évaluation de performances des systèmes informatiques**, Le Génie Logiciel et ses Applications, Toulouse, 4-8 Déc 1989.

[FER 78] D. Ferrari, **Computer Systems Performance Evaluation**, Prentice-Hall, 1978

[FDI 89] S. Fdida, G. Pujolle, **Modèles de Systèmes et de Réseaux - Tome 1: Performance**, Eyrolles, 1989.

[LES 88] H. Lesca, J. Peaucelle, **Elements d'informatique appliquée à la gestion**, Dalloz, 1988

[PUJ 89] G. Pujolle, M. Schwartz, **Reseaux locaux informatiques**, Eyrolles, 1989.

[QNA 90] QNAP2, Reference manual, version 7.0, Simulog, 1990.

[VAL 89] A. Valderruten, **Analyse des performances d'un systeme de communication**, document interne CERFIA/SIERA, 1989.

Distributed Implementation of a Linda Kernel *

Rogelio ALVEZ †
e-mail: d80403@buevm1.iinus1.ibm.com

Sergio YOVINE ‡
e-mail: sergio@granier.imag.fr

March 11, 1991

Abstract

A computer network is an interconnected collection of autonomous computers. A distributed system consists of a collection of concurrent, autonomous, descentralized and loosely coupled components. Linda is a high-level parallel programming language intended for distributed programming. It consists of a reduced set of primitives which can be embedded in any sequential language to transform it in a parallel one. This work describes a Linda kernel developed for a 3B Computer Network at ESLAI.

Contents

1 Introduction 2

2 Linda 3

2.1 Tuple Space . . . 3

2.2 Generative Communication . . . 4

2.3 Programming in Linda . . . 4

3 Operational Semantics of Generative Systems 6

3.1 Tuple Space . . . 6

3.2 Operational Semantics . . . 6

3.2.1 Syntax . . . 7

3.2.2 Semantics . . . 7

*This work was developed when the authors did the last year of their studies at Escuela Superior Latinoamericana de Informática (ESLAI)

†Computer Research and Advanced Applications Group. IBM Argentina S.A. (Piso 7) Ing. Butty 275 - 1300 Buenos Aires - Argentina.

‡LGI-IMAG. BP 53X - 38041 Grenoble - France.

4 Design alternatives 7

4.1 Distributing the Tuple Space 8

4.1.1 1-N distribution 8

4.1.2 N-1 distribution 8

5 Implementation 9

5.1 Partitioning the Tuple Space 9

5.2 Tuples and processes management 9

5.3 User's interface 11

5.4 Internal representation of tuples 11

6 Conclusions 12

7 Acknowledgements 12

A The C code 14

List of Tables

1 Distributed Array 5

2 The "Manager" process 10

3 in, rd and out operations 11

1 Introduction

A *computer network* [11] is an *interconnected* collection of *autonomous* computers, usually called nodes. Two machines are said to be interconnected if they are able to exchange information. By requiring a computer to be autonomous we mean that it has complete control of its functioning.

Computer networks make distributed applications feasible. A distributed system consists of a collection of concurrent, autonomous, descentralised and loosely coupled components.

Very often, network operating systems [12] do not provide high-level interprocess communication facilities. In fact, building distributed applications even if feasible, becomes a hard work which requires an expert knowledge of the low-level primitives of the network interface.

Linda [2,3,4,5,6] is a high-level parallel programming language intended for distributed programming. It consists of a reduced set of primitives over a virtual, associative shared memory,

called *tuple space*, which can be embedded in any sequential programming language to transform it in a parallel language.

Linda has some interesting features which make it adequate within a computer network environment. Firstly, in a distributed system services are required by name rather than by location [10]. Secondly, the operations of Linda can be embedded in any language. This allows, at least in principle, to implement the different components of a distributed application in different programming languages, choosing the more appropriate one for each task.

This work describes a Linda kernel developed for a 3B Computer Network[1] at ESLAI. The structure of this paper is as follows. In Section 2, we present the operations of Linda. We discuss the generative communication model and the programming style supported by Linda. In Section 3, we brefly study an operational semantics of Linda. In Section 4, we discuss several alternatives to design a run-time system for Linda in a computer network. In Section 5, we present the implementation developed.

2 Linda

Linda is a reduced set of simple operations intended for distributed and parallel programming [2,3,4,5,6]. Linda is based on a tuple space memory which supports generative communication.

2.1 Tuple Space

A tuple space is an associative memory which storage unit is the tuple. A tuple can be added, removed or read using the out, in or rd operations, respectively. Tuples are addressed via any collection of its values. This mechanism is called *structure naming*.

Linda has the following primitives:

- out(t) is an asynchronous operation that adds tuple t to the tuple space. The executing process continues immediately.
- in(s) is a synchronous operation that removes from the tuple space a tuple t matching template s. If many tuples match, only one of them is removed. If there is not any matching tuple, then the executing process waits for one to be added to the tuple space.
- rd(s) is the same as in(s), except that the tuple matching template s is not removed from the tuple space, but only read.
- eval(t) is the mechanism provided to create processes. It adds an *unevaluated* tuple to the tuple space and creates a process to evaluate it. Tuples created by eval are called active tuples, and play the role of processes in the generative communication model.

[1] 3B is a trade mark of AT&T.

Since the eval operation is quite complex and its semantics is not very clear, we will consider only the first three operations.

2.2 Generative Communication

Many concurrent processes encompassing a tuple space communicate by adding, removing and reading tuples. A process with data to communicate adds a tuple to the tuple space. A process requiring the data removes or reads the tuple. This mechanism is called *generative communication.*

Generative communication supports:

a. *time uncoupling*, in the sense that non coexisting processes can communicate via the tuple space (i.e. the tuple space is independent of the processes and always exist).

b. *space uncoupling*, since any process can read or remove tuples generated by any other process.

In fact, these properties are common to every shared-memory model of communication.

2.3 Programming in Linda

Linda supports a style of programming based on distributed data structures [6,4], that is data structures that can be accessed by many processes in parallel. The control of such a data structure is not centralized in a single process, but distributed among all those that use it. This property is called *distributed sharing*[2].

There are many examples of distributed data structures. For instance, the bag of tasks, which is the basis of the master-worker paradigm of parallel programming [3,4]. A bag can be easily implemented in Linda as a collection of tuples of the form ("bag",*Atask*). Any process can insert a task in the bag executing out("bag",*thistask*), or remove one from it via in("bag",var *newtask*).

Other interesting example is the distributed array. In Linda, an array is a collection of tuples of the form (*index,value*), and an additional tuple ("length",*integer-value*) to denote the length of the array. Assuming that the values range over the integers, an implementation of this data structure is shown in table 1.

This Linda implementation of a distributed array is simpler than the one presented in [13,14]. Also notice that the implementation of variable length arrays is straightforward.

```
proc create(n: integer);
(* to create an array of length n *)
(* 'any' is a special value matching all types *)
     var i: integer;
     begin
          for i:=1 to n do out(i,any);
          out("length",n)
     end

proc put(i,v: integer);
(* to assign value v to element i, provided 1≤i≤n *)
     var j: integer;
     begin
          in(i,var j);
          out(i,v)
     end

proc get(i: integer): integer;
(* to read the value of element i *)
     var v: integer;
     begin
          rd(i,var v);
          return v
     end

proc length(): integer;
(* to know the length of the array *)
     var l:integer;
     begin
          rd("length",var l);
          return l
     end
```

Table 1: Distributed Array

3 Operational Semantics of Generative Systems

A *generative system* consits of a tuple space and a set of concurrent processes performing in, out and rd operations.

The purpose of this section is to describe from an *observational* point of view, the behavior of a generative system.

3.1 Tuple Space

Tuples are the basic elements of information within the shared-memory model of generative communication. In the previous section we have defined informally how a template and a tuple match in the tuple space. We will give a formal description of this operation now.

Let $\mathcal{S} = \{s_1, ..., s_n\}$ be a set of sorts, $\mathcal{C}_{s_i}$ the set of constants of sort s_i, $\mathcal{C} = \bigcup_{i=1}^{n} \mathcal{C}_{s_i}$, and $\mathcal{X}$ a set of variables.

Definition 3.1.1 *Tuple.*
A tuple is defined as $t = (t_1, ..., t_m)$ *where:*
$(\forall i \in \{1, ..., m\})$
i. $t_i = x : s, x \in X \wedge s \in \mathcal{S}$, *or*
ii. $t_i = c : s, c \in \mathcal{C}_s$.

Definition 3.1.2 *Matching.*
Let $t_1 = (x_1 : s_1, ..., x_m : s_m)$ *and* $t_2 = (y_1 : r_1, ..., y_m : r_m)$ *be two tuples. Then, a predicate match*(t_1,t_2) *is defined as follows:*
$match((x_1 : s_1, ..., x_m : s_m), (y_1 : r_1, ..., y_m : r_m))$
$\quad \equiv_{def} (m = k) \wedge (\forall i \in \{1, ..., m\} : match_comp(x_i : s_i, y_i : r_i))$
$match_comp(x : s, y : r)$
$\quad \equiv_{def} (s = r) \wedge ((x \in \mathcal{C}_s \wedge y \in \mathcal{C}_r \wedge x = y) \vee (x \in \mathcal{C}_s \wedge y \in \mathcal{X}) \vee (x \in \mathcal{X} \wedge y \in \mathcal{C}_r))$

Definition 3.1.3 *Tuple Space.*
A Tuple Space is a multiset of tuples. We will use $\oplus$ *to denote the union of multisets.*

3.2 Operational Semantics

In order to modelize generative systems, we will define a simple language whose terms denote sets of parallel processes performing the operations of Linda. Then, we will define an operational semantics for this language by means of a transition relation on terms [1].

3.2.1 Syntax

The syntax is as follows:
$A := in(x) \mid out(x) \mid rd(x)$
$E := nil \mid A.E$
$P := E \mid P||P$

3.2.2 Semantics

Let $q, q', p, p' \in P$ and $\varepsilon, \varepsilon'$ be tuple spaces. We define a transition relation $\rightarrow$ as follows:

(1) $$(out(x).p, \varepsilon) \xrightarrow{out} (p, \varepsilon \oplus \{x\})$$

(2) $$\frac{match(x,y)}{(in(y).p, \varepsilon \oplus \{x\}) \xrightarrow{in} (p, \varepsilon)}$$

(3) $$\frac{match(x,y)}{(rd(y).p, \varepsilon \oplus \{x\}) \xrightarrow{rd} (p, \varepsilon \oplus \{x\})}$$

(4) $$\frac{(p, \varepsilon) \xrightarrow{\alpha} (p', \varepsilon')}{(p||q, \varepsilon) \xrightarrow{\alpha} (p'||q, \varepsilon')}$$

(5) $$\frac{(q, \varepsilon) \xrightarrow{\alpha} (q', \varepsilon')}{(p||q, \varepsilon) \xrightarrow{\alpha} (p||q', \varepsilon')}$$

Finally, we introduce *fairness* as an axiom.

4 Design alternatives

There are two important questions that naturally arise when we want to design a Linda virtual machine [2]:

- How to find the tuples?
- How to store them?

The first is closely related with the criterion adopted to distribute tuple space; the second is related with the choice of required data structures to maintain it.

4.1 Distributing the Tuple Space

Since tuple space is logically similar to a shared memory, the task of building and maintaining it over disjoint-memory hardware - like a local network - represents an interesting implementation problem.

We should place tuple space completely in one node and direct all generative operations to it, but this policy might generate a dangerous "bottleneck", degrading the performance. Furthermore, this solution is not conceptually the most adequate if we want to have an authentic distributed environment. It seems quite clear that we should distribute tuple space in some way over some subset of all nodes. There are several forms to distribute the tuple space. In general, they consist of partitioning the space into small subsets, assigning the managment of each one to a group of nodes. We will brefly discuss here only two extreme cases, in which single nodes are considered instead of groups.

4.1.1 1-N distribution

One possible choice is the following: when a process executes an out(t) operation, the Linda kernel sends a copy of tuple t to every node in the network. Thus, when a process wants a tuple, it only needs to search in its node, because tuple space is completelly replicated in each node. But the attempt to delete it (by means of an in operation) implies the activation of a global protocol, because the tuple must be deleted in all nodes [8]. This option is viable with good performance only if we have great memory capacity and fast communication medium, mainly with reliable broadcast.

4.1.2 N-1 distribution

The inverse choice is also possible. The tuple space can be partitioned into N small parts, each one assigned to one of the nodes of the network. There are several criteria to partition the tuple space:

- *by locality*, tuples are located where they are produced. The out operation is local, while in and rd may cause searching all over the network if the tuple is not locally available. Deleting tuples becomes easier [3]. However, this mechanism could produce overloading of certain nodes, specially those where out operations are very often performed, if not controlled.

- *by a distribution function*, tuples are located according to the value produced by applicating it a hash function. This mechanism can control overloading [9].

In this case, tuples are not replicated. So, its goal is to make an efficient managment of space.

5 Implementation

In this section we will present the Linda kernel implemented for the 3B computer network at ESLAI, consisting of two nodes interconnected via Fthernet.

5.1 Partitioning the Tuple Space

The available hardware did not have enough memory to adopt the 1-N policy discussed in Section 4. Thus, we chose the N-1 alternative with a distribution function criterion. We implemented the tuple space as a great hash table, distributed throughout the network. This table was partitioned in subtables, actually implemented as hash tables, one in each node. On executing an out(t) operation, the Linda kernel will send tuple t to the unique node determined by the application of the hash function to t. Something similar occurs when a tuple s is desired (by means of an in or rd operation). We must ask for that tuple to the node determined by the result of the application of the hash function to s.

Notice that this last policy makes an efficient use of memory, because information is not replicated. Besides, it also minimizes communication, because it is neccesary only one step of communication for an out operation and two steps for in and rd operations (ask and reply).

5.2 Tuples and processes management

The subspace installed in each node is managed by an "endless" process, independent of the application. This process was called "Manager" (table 2).

Manager is encouraged to assign a particular tuple (generated by an out) to a template (generated by an in or rd). The applications' requirements are coded in messages and transmited through the network. Each message includes the operation type, the tuple and the identifier of the requesting process. The message is sended to the node determined by the "space-partition" function (see table 3).

When a process executes an in or rd operation, it sends the request by a message and blocks waiting for the reply. Each in or rd may or not find a matching tuple. In the first case, the manager replies inmediatelly with the found tuple. In the latter, the manager stores the message until a matching tuple arrives. When a new tuple arrives (via out), the manager searchs for delayed matching templates waiting for that tuple. If no messages containing matching templates are found, the tuple is stored in the local space; otherwise the messages are replied. This strategy prevents starvation.

```
proc Manager();
    var te: Tuple_space;
    var se, s: set of Mge;
    var t1, t2: Tuple;
    var op1, op2: Opr;
    var pid1, pid2: ProcId;
    loop forever
        receive((pid1,op1,t1));
        case op1 do
        in:  if ∃t ∈ te such that match(t,t1)
             then te:=te ⊖ t; send((Manager,_,t),pid1);
             else es:=es ⊕ (pid1,op1,t1);
        rd:  if ∃t ∈ te such that match(t,t1)
             then send((Manager,_,t),pid1);
             else es:=es ⊕ (pid1,op1,t1);
        out:
             s:=set of mges in es waiting for t1;
             es:=es ⊖ s;
             for (pid2,rd,t2) ∈ s do send((Manager,_,t1),pid2);
             if ∃ (pid2,in,t2) ∈ s
             (* there is at most one *)
             then send((Manager,_,t1),pid2);
             else et:=et ⊕ t1;
        end case
    end loop
end proc
```

Table 2: The "Manager" process

```
type Opr = enum  in, rd, out ;
type Mge = record pi: ProcId; op: Opr, t: Tuple end;

in(t):
     send((process_id,in,t),dest(t));
     receive((Manager,_,t'));
     instantiate(t,t')

rd(t):
     send((process_id,rd,t),dest(t));
     receive((Manager,_,t'));
     instantiate(t,t')

out(t):
     send((process_id,out,t),dest(t))
```

Table 3: in, rd and out operations

5.3 User's interface

A library implementing the operations of Linda is provided. It can be included in any C program. The operations have a list of parameters. The first element of the list is a string defining the type of the tuple. The rest of the list contains the tuple. Each element of a tuple can be of type string (indicated by an "s"), integer ("i") or double ("f"). Formal parameters (i.e. variables), must be preceded by a symbol "&". For instance, in("sii&i","pepe",1,1,&val), corresponds to an invocation of the in operation with a template of length four, which first argument is the string "pepe", the second and the third are the integer 1, and the last component is a formal argument of type integer. The variable "val", must be defined in the program performing the operation.

5.4 Internal representation of tuples

A tuple is implemented as a structure containing the length of the tuple and an array of components. In fact, the length of the tuples has a maximum. This decision was adopted in order to be able to transmit tuples in a single Ethernet packet (up to 1500 bytes [11]).

Each component has three fields: its type, if it is a variable or a value, and its value (which is undefined if it is a variable).

6 Conclusions

The design of a run time support must include not only the tuples but also the processes that are communicating via tuple space. Since we did not consider active tuples, we decided to exploit the multiprogramming environment provided by UNIX. Thus, we designed the run time support in order to let Linda processes be in two coupling levels simultaneously: multiple processes running in one node and multiple nodes (in practice just two) communicating via Ethernet. Linda permits an uncoupled programming style. Therefore, it is not necessary to determine a priori the number of processes that participate in any application. Whatever is the number of them, they can be created at any time and in any of both nodes; there exist just hardware limitations.

"Manager" processes are installed at the system "start-up" time. The entire system has been implemented in C. For network management we have used the network interface provided by the operating system.

The kernel is totally independent of the number of nodes in the network. Nowadays, it is working with two nodes, but it has been designed to work in the same way with one node or more. It is only necessary to actualize a special system table when the hardware configuration changes.

In the implemented scheme, all tuple requirements involve the use of the communication channel, even though the process that requires a tuple and the manager of the subtable where it is placed are in the same node. These kind of requests could be avoided if tuple subspaces in each node were shared by their managers and local processes. The main advantage is that in this new scheme each manager is only responsible for the administration of external requests. However, the manager and the local processes must compite for the use of the subspace. We believe that this scheme will substantially improve the performance of the applications, because of the reduction in the use of the communication bus. This improvement can be achieved using the IPC facilities (shared memory, semaphores and messages) of UNIX.

7 Acknowledgements

We would like to thank Nick Carriero, who sent us a lot of papers about Linda, and for his suggestions and advise. Daniel Yankelevich made important criticisms and suggestions. Jorge Aguirre gave us his assitance and direction.

References

[1] R. Milner. "Communication and Concurrency". Prentice Hall Inc., C.A.R. Hoare, Series Editor, 1989.

[2] D. Gelernter. "Generative Communication in Linda". ACM Transactions on Programming Languages and Systems, pages 80-112, January 1985.

[3] S. Ahuja, N. Carriero, D. Gelernter. "Linda and Friends". IEEE Computer, Aug 86, 26-34.

[4] N. Carriero, D. Gelernter. "How to write parallel programs: A guide to the perplexed". Yale University, Dept. of Computer Science, Nov. 88 (to appear, ACM Computer Surveys).

[5] N. Carriero, D. Gelernter. "Linda in context". (to appear, Communications ACM).

[6] D. Gelernter, N. Carriero, S. Chandran, S. Chang. "Parallel programming in Linda". *Proceedings of the International Conference on Parallel Processing*, Aug. 85, 255-263.

[7] Nicholas J. Carriero. "Implementing Tuple Space Machines". PhD thesis, Yale University, New Haven, Connecticut, 1987. Department of Computer Science.

[8] N. Carriero, D. Gelernter. "The S/Net's Linda Kernel". *Proceedings Symp. Operating Systems Principles*, Dec. 85.

[9] R. Bjornson, N. Carriero, D. Gelernter. "The implementation an Performance of Hypercube Linda". Research report YALEU/DCS/RR-690, Mar. 89.

[10] Andrew S. Tanenbaum. "Computer Networks". Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981.

[11] Andrew S. Tanenbaum. "Computer Networks". Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1989.

[12] A. Tanenbaum, R. Van Renesse. "Distributed Operating Systems". Computing Surveys, 17(4), Dec. 85, 419-470.

[13] W. Horwat, A. Chien, W. Dally. "Experience with CST: Programming and Implementation".

[14] W. Dally, A. Chien. "Object-Oriented Concurrent Programming in CST". SIGPLAN NOTICES 24(4), Apr. 89.

A The C code

We include here the main parts of the implementation.

```
/* red.h */
#include <sys/ni.h>
#include <fcntl.h>

#define NID6CPY(a,b)  (a[0]=b[0],\
                       a[1]=b[1],\
                       a[2]=b[2],\
                       a[3]=b[3],\
                       a[4]=b[4],\
                       a[5]=b[5])

#define NID2CPY(a,b)  (a[0]=b[0],\
                       a[1]=b[1])

#define PK_SIZE 250          /* Packet Length*/

#define NINETPORT "/dev/ni"  /* Network Interface driver of 3B */

typedef char id_port[6];     /* port id */

typedef struct { EI_PORT head;
                 char    data[MAXETPACKET];
               } PACK ;

static id_port ELENA_PORT = {0x69,0x69,0x69,0x69,0x69,0x69} ;

static id_port ESLAI_PORT = {0x71,0x71,0x71,0x71,0x71,0x71} ;

static char APPROT[2] = {0x88,0x88} ;

static short approt = 0x8888;

static int RdPORT;     /* Read Port */

static int WrPORT;     /* Write Port */

static int ErrPORT;    /* Error code for network operations */

static id_port MyPORT;
```

```
/*-------------------------------------------------*/
/*                    MANAGER                      */
#include <stdio.h>
#include "red.h"
#include "mensajes.h"
#include "hash.h"

#define TRUE 1
#define FALSE 0

extern int errno;

main()
{
   mensaje *m,*ReadPORT();

   if (InicET()==-1) {
      printf("ERROR %d InicET() \n",errno);
      exit(1);
   }

   InicKernelRdPORT();
   if (ErrPORT) {
      printf("ERROR %d InicKernelRdPORT() \n",ErrPORT);
      exit(2);
   }

   InicKernelWrPORT();
   if (ErrPORT) {
      printf("ERROR %d InicKernelWrPORT() \n",ErrPORT);
      exit(3);
   }

   while (TRUE) {
      if (!(m=ReadPORT())) {
         printf("ERROR %d ReadPORT() \n",ErrPORT);
         exit(4);
      }
      switch(m->op) {
         case IN : Gin(m);
                   break;
         case OUT: Gout(m);
                   break;
         case RD : Grd(m);
```

```
                    break;
            }
        }
    }

    /* rd() operation */
    Grd(p_mje)
        mensaje *p_mje;
    {
        int h_entry;
        LDT *LDTcte;
        int escape=FALSE;
        mensaje *m,*ConstMJE();

        h_entry = hash(p_mje);
        LDTcte=ET[h_entry].lt;
        while ( !escape && LDTcte->sig ) {
            LDTcte = LDTcte->sig;
            if (match(*(p_mje->t),*(LDTcte->t))) {
                m=ConstMJE(MyPORT,NIL,LDTcte->t);
                WritePORT(m,p_mje->origen);
                free(m);
                escape = TRUE;
            }
        }
        if (escape)
            BorrarMJE(p_mje);
        else
            ET[h_entry].lm.ult=AgregarLDM(ET[h_entry].lm.ult,p_mje);
    }

    /* in() operation */
    Gin(p_mje)
        mensaje *p_mje;
    { ... }

    /* out() operation */
    Gout(p_mje)
        mensaje *p_mje;
    { ... }

    /* ------------------------------------------------------*/
    /*                    Network Management                 */
```

```
mensaje *ReadPORT()
{
   PACK Pac_In;
   mensaje *PackToMen();

   ErrPORT = 0;
   if (read(RdPORT,&Pac_In,PK_SIZE+2)==-1) {
      ErrPORT = errno;
      return NULL;
   }
   return PackToMen(Pac_In);
}

WritePORT(m,nodo)
   mensaje *m;
   char *nodo;
{
   PACK *MenToPack();

   return write(WrPORT,MenToPack(m,nodo),PK_SIZE);
}

InicKernelRdPORT()
{
   NI_PORT vpst;
   int fd;
   void KernelPORT();

   ErrPORT = 0;
   if ((fd=open(NINETPORT,O_RDONLY))==-1)
      ErrPORT=errno;
   if (ioctl(fd,NIGETA,&vpst)==-1)
      ErrPORT=errno;
   KernelPORT(vpst.srcaddr);
   vpst.rspq_sz = 8;
   vpst.rcvb_sz = PK_SIZE+2;
   vpst.rcvq_sz = 15;
   vpst.protocol = approt;
   vpst.type = ETHERTYPE;
   if (ioctl(fd,NISETA,&vpst)==-1)
      ErrPORT=errno;
   RdPORT = fd;
}
```

```
InicKernelWrPORT()
{ ... }

void ProcessPORT(str)
char *str;
{
   register int i,j;
   int  id;
   id_port pid;

   sprintf(pid,"%d",getpid());
   str[0] = 0x69;
   str[1] = str[4];
   str[2] = str[5];
   for(i=strlen(pid);i<6;i++)
      pid[i]='1';
   j = 3;
   for(i=0;i<3;i++) {
      sscanf(&pid[2*i],"%2x",&id);
      str[j++] = id;
   }
}

void KernelPORT(str)
char *str;
{
   if (str[5] == 0x9d )
      NID6CPY(str,ELENA_PORT);
   else
      NID6CPY(str,ESLAI_PORT);
}


/*-------------------------------------------------------------*/
/*                              IN                             */
in(va_alist)
va_dcl
{
    mensaje *m,*ConstMJE(),*ReadPort();
    va_list ap;
    char *GetVsort(),*MyPort(),
         *signatura,
         *str;
    dir d[TUPLAMAX];
```

```
  vsorts vsort_cte;
  int i=0;
  int lg;
  tupla *tup;

  tup=(tupla *)malloc(sizeof(tupla));
  va_start(ap);
  signatura = va_arg(ap,char *);
  while (*signatura==' ') signatura++;
  while (*signatura != '\0') {
        signatura = GetVsort(signatura,&vsort_cte);
        switch(vsort_cte) {
              case VALINT: { ... }
              case VALDBL: { ... }
              case VALSTR: { ... }
              case VARINT: { ... }
              case VARDBL: { ... }
              case VARSTR: { ... }
              case V_ERR  : return -1;
        }
        i++;
  }
  tup->lg=i;

m=ConstMJE(MyPort(),IN,tup);
if (WritePort(m,DestPort(tup->comp[0].campo.s))==-1) {
   BorrarMJE(m);
   return -1;
}
else
   BorrarMJE(m);
if (!(m=ReadPort())) {
   BorrarMJE(m);
   return -1;
}
/* Instantiation of formal parameters */
for (i=0;i<m->t->lg;i++)
    switch(m->t->comp[i].sort) {
          case INT : if (d[i].fi != NULL)
                        *(d[i].fi)=m->t->comp[i].campo.i;
                     break;
          case DBL : if (d[i].ff != NULL)
                        *(d[i].ff)=m->t->comp[i].campo.f;
                     break;
```

```
            case STR : if (d[i].fs != NULL)
                          strcpy(d[i].fs,m->t->comp[i].campo.s);
                       break;
        }
    BorrarMJE(m);
    return 0;
}
```

Programming with Structures, Functions, and Objects*

Fritz Henglein
DIKU
University of Copenhagen
Universitetsparken 1
DK-2100 Copenhagen
Denmark
henglein@diku.dk

Konstantin Läufer
Courant Institute of Mathematical Sciences
New York University
715 Broadway, 7th floor
New York, NY 10003
USA
laufer@cs.nyu.edu

March 18, 1991

Abstract

We describe program structuring mechanisms for integrating algebraic, functional and object-oriented programming in a single framework. Our language is a statically typed higher-order language with specifications, structures, types, and values, and with universal and existential abstraction over structures, types, and values.

We show that existential types over structures generalize both the necessarily homogeneous type classes of Haskell and the necessarily heterogeneous object classes of object-oriented programming languages such as C++ or Eiffel. Following recent work on ML we provide separate linguistic mechanisms for reusing specifications and structures. Subtyping is provided in the form of explicit type conversions.

The language mechanisms are introduced by examples to emphasize their pragmatic aspects. We compare them with the mechanisms of XML+, Haskell and Eiffel and give a type-theoretic perspective. These mechanisms have been developed within a larger, ongoing prototyping language design project.

*Supported by the Defense Advanced Research Projects Agency/Information Systems Technology Office under the Office of Naval Research contract N00014-90-J-1110

1 Introduction

We describe program structuring mechanisms for a high-level language currently under development as part of a software prototyping research effort. Our main aim is to support and integrate algebraic, functional, and object-oriented programming in a coherent language framework.

The language concepts described in this paper are intended to serve as a platform for the design of the prototyping language Griffin [D+90] currently under development at New York University.

Our linguistic mechanisms can be seen as a combination of Ada (generic) packages [Uni83], ML polymorphism, signatures and structures [Har86], Eiffel object classes [Mey88], Haskell type classes [HW90], and (partially) abstract types [MP88]. Our design is most closely related to the recently proposed extension of Standard ML with subtyping and inheritance [MMM91].

We distinguish between three levels of "entities" in our language: *specifications* at the highest level, *types* and *structures* in the middle, and *values* at the bottom.[1] Specifications consist of *interface* and *constraint* specifications. Specifications are inhabited by structures, which consist of zero, one, or more types and operations on those types. A type, in turn, denotes a set of values. Types, in some sense, pre-exist values: Every value has a type and comes into existence if and when its type is defined, typically as part of a structure definition. This may be at language design time (for built-in types) or at program design time (for user-defined types). A specification corresponds to its use in algebraic specification [EM85]; in object-oriented terms it can be roughly equated to the (public) interface of a class along with its invariants and assertions (see, e.g., Eiffel [Mey88]). In algebraic terms a structure is a valid model of a specification, and the types in a structure correspond to the carrier sets of such a model; in object-oriented terms the types in a structure represent the data part of a class implementation, and the operations stand for the methods. Detailed comparisons with other languages are given in Section 2.

We distinguish between the *explicit* language and the *implicit* language. Static and dynamic semantics are specified only for the explicit language (c.f. [MH88]). The programmer may, however, write in an abbreviated style relying on the programming environment to fill in missing information, interacting with it, if necessary. Thus the task of completing an implicit program can be seen as a purely combinatorial task that does not have to be supported in a uniform fashion in all programming environments since it is outside the language definition proper. Several pieces of information may be inferred by a programming environment:

- "classical" implicit type information, which models universal polymorphism;,
- applications of conversion functions, which models subtyping (c.f. [BCGS89]);
- structure resolving, which models static and dynamic overload resolution.

Whereas the type-theoretic ideas in our approach have appeared in various forms before, their combination and application to object-oriented programming appear to be new.

[1] Actually there is an even higher level, *modules*, that contain bindings for entities of all lower levels.

Instead of implicit subtyping we may define explicit conversion functions between types; for example, we may define a conversion from integers to reals and vice versa, or a conversion from cartesian colored points to cartesian points. This allows subtyping even for types defined as part of structures that do not satisfy the rule of [CCH+89], which has been criticized by Meyer as too restrictive [Mey89] and has been adopted in an even more restrictive fashion in [MMM91].

As pointed out before, a class in the object-oriented sense is modeled by a specification for a single type and, by extension, all structures that satisfy it, either by declaration or implicitly. The type of a "generic" object of such a class is represented by the existential type $\exists s : SPEC.\,|s|$ where $|s|$ denotes the type in structure s. An element of this type has two components, a structure s that satisfies specification $SPEC$ and a value of type $|s|$. This is an important difference from the approach taken in XML+ [MMM91]: there an object has also two components, a type component and a value component. The value component consists of all data *and* all their methods. So for two XML+ objects we cannot be sure whether their methods or their data representations are identical. In particular, binary methods that operate on objects with identical representation type are problematic in that approach.

Dynamic binding can be achieved by using the structure component of a generic object and selecting from it the corresponding operation and applying it to the value part of an object. For example, $x.f(w)$ in a language such as Eiffel is translated into

$$\text{let } < s : SPEC, v : |s| >= x \text{ in } s.f(v, w)$$

or

$$|x\,|\,.f(val(x), w)$$

where x is a generic object consisting of a structure s and a value v. In the implicit language we could have written $f(x, w)$ with the idea that the conversion *val* and the structure modifier $|x|$ can be inferred. Inferring structure modifiers of this sort is generally *dynamic overload resolution*; if the structure modifier is a (compile time) constant structure then it is essentially *static overload resolution.*

Specifications and structures are separate, as in XML+. This permits specifications with no structures or more than one structure, obviating the need for deferred classes and artificial subclasses. Also, specifications and structures can be independently reused — "inherited" in object-oriented lingo. This permits an independent development of specification enhancement and implementation specialization (c.f., [Sny87]).

Functional abstraction over structures of a specification makes it possible to treat members of an object class as first-class functions. This is possible because class members in our language do not have implicit arguments and because the range of applicability can be precisely described by a specification. For example, the method "translate" of object class Point has type

$$\text{translate}[s : Point](p : |s|, x, y : real) : |s|$$

The paper is organized as follows: Section 2 gives an overview of problems occurring in existing object-oriented languages, the functional language Haskell, and XML+. Section 3 introduces the key features of our language, called G in the remaining text. Section 4 describes the type-theoretic foundations of G along the lines of XML [MH88] and XML+ [MMM91]. We conclude with a summary of the contributions made and an outlook on further research. Appendix A contains a collection of program examples in G.

2 Some Problems of Other Approaches

2.1 Object-oriented Languages

Most strongly-typed object-oriented languages [Mey88,Str86] identify inheritance with subtyping, where the subtyping is based on extensible record types partially ordered with respect to a subtyping relation. This view appears too restrictive when it comes to modeling certain algebraic structures. Common properties of classes are typically factored out in a common superclass, so that heterogeneous structures can be constructed. To illustrate this, let us consider the following example. We define a class of partially ordered objects with the following signature:

```
class PartialOrder is
        less: PartialOrder -> Bool
```

Now we refine `PartialOrder` to a class `Int` with an addition operation, as in

```
class Int superclass PartialOrder is
        less: Int -> Bool
```

However, the definition of method `less` in class `Int` violates the contravariance rule for function subtyping. Hence it is not possible to define `Int` as a subtype of `PartialOrder`. This example shows how the requirement that inherited classes be subtypes of their superclasses guarantees type safety but inhibits flexibility.

Trading off in the other direction, Eiffel abandons the contravariance rule in its type system and gives up static type safety for the sake of greater flexibility. Therefore we can actually create a class `PartialOrder` of partially ordered objects and several subclasses of `PartialOrder`, e. g. `Int` and `String`. Indeed, the Eiffel model allows us to write the following code:

```
bool compare(PartialOrder x, y)
        return x.less(y);
n = Int(3);
s = String("three");
n.compare(s)
```

which would lead to a runtime type error.

Another weakness of object-oriented languages is the identification of specification and implementation in a single construct. (An exception is the language Emerald [BHJL86], which separates specification and implementation, but lacks a reuse mechanism.) While the specification and the

ıplementation of a class may be syntactically separate in order to support modular coding, it is ɔt possible to identify a single specification with *multiple implementations.* If we tried modeling ıe specification as a common superclass and the implementations as subclasses, we would run into vo problems. The first one is the contravariance pr-ɔblem mentioned above. The second problem ıs to do with the *interoperability* of objects of the same specification, but different implementation. s an example, consider the problem of providing a hash table and a linked list implementation of set. The problem appears in coding binary operations such as "union" so that they work on a air of hash tables or a pair of linked list representations, but not necessarily on mixed pairs. In ı object-oriented language union will always have to work on mixed pairs as well.

.2 Haskell Type Classes

askell ([HW90]) provides an overloading mechanism ([WB89]) which is different from the subtyp- ıg found in traditional object-oriented languages and solves the problem described above. Instead f capturing common properties of several classes by deriving them from a common superclass, in askell such classes are explicitly declared to be instances of the same *type class.* Unlike a class in bject-oriented languages, a type class is not a type itself. Instead, it specifies certain properties equired of its instance types.

In Haskell, we could express the orderedness of type `Int` by defining `PartialOrder` as a type lass, and declaring `Int` and possibly other types as instances of `PartialOrder`. We are allowed to onstruct hierarchies of type classes. We may define e. g. a type class `Num` with numerical operations s a subclass of `PartialOrder`. `Int` could then be made an instance of `Num`, along with other types uch as `Float`.

Nevertheless, the Haskell model has several major shortcomings. First, Haskell does not provide mechanism for inheritance at the implementation level. Each instance of a type class has to be mplemented without reusing other implementations. Second, Haskell's type classes cannot be arametrized nor used as parameters of type constructors. For example, it is not possible to define list over the type class `Num` whose members could be of any type declared as an instance of `Num`. Neither is it possible to define a type class `Set` parametrized by the element type. See [Ode90,OL91] or a detailed treatment.

.3 XML+

Various disadvantages of existing object-oriented languages have been recognized and described ı [Mit90b,MMM91]. XML+ is an extension of the Standard ML module system and improves ver both the object-oriented and the Haskell style in a number of respects. A major shortcoming f object-oriented languages is the merging of specification and implementation of classes. One onsequence of this merging is the difficulty of providing multiple implementations of the same class pecification. Another problem stems from the observation that specification and implementation ıheritance (extension) often work most naturally in opposite ways. Consider e. g. queues and tacks as seen in section A. It is natural to specify a queue as an extension of the specification of

a stack, since a queue provides at least all the operations that are part of a stack. On the oth hand, a stack implementation can easily be obtained from a queue implementation by hiding tl operations that are not needed. However, we lose this abstraction if we implement a queue in tern of a stack, because the queue implementation would have to know the representation of the sta to provide the additional operations [Sny87].

XML+ separates specification and implementation, using an extension of ML signatures f specifications, and an extension of ML structures for implementations, combined with separate i heritance mechanisms. Separate mechanisms for specification and structure subtyping are provide F-bounded polymorphism ([CHC90]) is used to allow polymorphism over families of structurall similar types of objects that do not necessarily have a subtyping relationship. Furthermore, XML structures support traditional *abstract data types*, i. e. pairs consisting of a representation typ and a set of operations on that type, which are not present in existing object-oriented language XML+ introduces *internal interfaces* which are used by multiple implementations to interact wit one another.

We have found some short-comings in the XML+ support for object-oriented programmin which we will illustrate below. First, we find it important to be able to specify that two distinc objects have identical representation type and methods. In existing object-oriented languages suc as C++ it is normally the case that two instances of the same class differ semantically only in the state. XML+, however, lacks a mechanism to guarantee this, as illustrated by an example:

```
specification OBJ = spec
                      val get: unit -> int;
                      val put: int -> unit
                    end
structure Obj1: OBJ = struct
                        val i: ref int = ref 0;
                        fun get () = !i;
                        fun put k = i := k
                      end
structure Obj2: OBJ = struct
                        val i: ref int = ref 2;
                        fun get () = 2 * !i;
                        fun put k = i := k + 3
                      end
```

Given specification `OBJ`, we can declare objects that are instances of `OBJ` and therefore satisfy th signature requirements. However, we are free to implement the components of the instances i different ways as long as the signature is correct. There is no direct mechanism to construct severa instances of the same implementation; it can be approximated using code reuse at structure level

G provides specifications, structures, and objects. While specifications contain signature infor mation, structures implement representation and operations of a class or an ADT. By creatin several instances of the same structure, we are sure that they are identical and only differ in thei state.

The second problem stems from the inheritance mechanism for structures and appears in

number of object-oriented languages. Inheritance in XML+ can be described as a textual copying combined with a renaming and visibility control mechanism. Consider the following example:

```
structure Amount = struct
                     type t = int * int;
                     val v: t = (12, 95)
                   end
structure UseAmount = struct
                        copy Amount;
                        fun dollars () = #1 v;
                        fun pennies () = #2 v
                      end
```

This textual copy mechanism is not safe; it may lead to problems known from other languages such as Smalltalk [GR83]. The problem generally appears in the part of the code that is copying from the other structure. If `Amount` is redefined with the representation given below, a type error occurs in `UseAmount`.

```
structure Amount = struct
                     type t = int;
                     val v = 1295
                   end
```

At specification level, besides textual copying, XML+ provides extension and restriction of specifications, which result in subtypes and supertypes of the original specification, respectively. Unfortunately, it is not clear how a recursive specification is extended. By a recursive specification we mean one whose name appears in the signature of one of its components. Suppose we extend the specification

```
specification StackClass[type t] = spec
                                     fun push: t -> StackClass[t]
```

to another specification

```
specification QueueClass[type t] = extend StackClass[t] with
                                     fun enqueue: t -> QueueClass[t]
```

Does the component `push` of `QueueClass` have type `t -> StackClass[t]` or `t -> QueueClass[t]`? Clearly, the latter would be more desirable as we might want to push an element onto our queue first, and then enqueue another one; hence we want `push` to return a queue. Traditional object-oriented languages provide constructs such as `mytype` or `like current`, while *G* solves this problem by identifying the object type with the representation type, and not with the type of the whole structure.

Although XML+ contributes to the solution of various problems present in the object-oriented paradigm, it does not completely resolve certain issues regarding code reuse and class specification.

3 Specifications, Types, and Values

In this section, we will present the data abstraction mechanisms of our language *G*. Let us give an overview of the concepts and terms we use, before we address the technical issues.

3.1 Basic Concepts and Terminology

type: A type denotes a collection of (first-class) values.[2] A type is not useful by itself; it is normally defined as a component of a structure, which provides operations involving that type. Types are either primitive, such as `int`, `bool`, `string`, or constructed from primitive types, such as function types, ML-style datatypes, etc. In G, monomorphic and polymorphic functions are considered first-class.

value: A value is an inhabitant of a type. Values are exactly the first-class entities in G, i. e. they can be returned as the result of a conditional expression, passed as a function parameter, or stored in a variable. Examples are `3`, `false`, `fn[t :: Type] (x : t) => x`.

structure: Structures in G are viewed the same way as in ML. A structure is an encapsulation unit that consists of zero, one, or more types, and zero, one, or more values. Structures inhabit specifications; although we could say that specifications are types of structures, we prefer to use the term "type" exclusively for types of first-class values. Structures containing one or more types are not first-class entities, since that would inhibit static typing. They are at the same level as types; in fact, a type may be identified with a structure containing only that (representation) type, but no operations. All predefined types are actually representation types of structures that define the operations on them.

functor: A functor is a structure template parametrized by (compile-time) values, types, or structures. Functors are inhabitants of parametrized specifications.

specification: A specification defines the interface of a structure, i. e. its visible components.[3] A structure is said to inhabit, or *implement* a specification if it provides definitions for all the entities required by the specification. Specifications can be parametrized by values, types, or structures; in that case, they are specifications of functors. So far, our specifications correspond to ML signatures or Ada package specifications. Furthermore, specifications can be parametrized with respect to other signatures.

module: A module is a compilation or library unit that contains bindings for entities of all lower levels, i. e. specifications, structures, types, and values.

hidden type and generic object: In many situations, we want to have objects that are instances of some implementation of a specification, but we do not care which one. Such *generic* objects consist of a (hidden) structure component and the object value, whose type is the representation type of the structure component.[4] They provide dynamic dispatching on subclasses in the sense that the structure component is hidden and may be locally opened in

[2] Hence in type-theoretic terms, types in G are "small" types. See Section 4 for a brief discussion of polymorphic functions.

[3] Optionally a specification can also contain constraints, which are, however, of no further relevance here.

[4] Such hidden types are called existential types in type-theoretic terminology, see [MP88]

order to access the methods implemented on that particular representation type. The value component can be accessed only within an open statement; this restriction allows us to treat hidden types as first-class types.

reuse: Code reuse, or inheritance, can occur both at structure level and specification level. Reuse at structure level is not yet well-understood (see the discussion in [Mit90b]), and is currently handled by textual copying, although we are not satisfied with this method. At specification level, new specifications can be derived from previous ones by copying, adding, or omitting parts of the specification.

subtyping and conformity: At type level, *G* provides no implicit subtyping; instead, explicit conversion functions may be used. This gives us higher flexibility for forms of subtyping that do not satisfy the restrictive contravariance rule for record subtyping [CHC90]. On the other hand, we have implicit conformity between specifications; for example, when a parameter is specified by a required interface, any structure that satisfies that interface may be passed.

3.2 Specification and Implementation

Let us now demonstrate how types can be specified and implemented. A specification states the abstract properties of a type, i. e. how a type will be used, but does not specify any concrete implementation details (except possibly for representation-independent bodies of operations, as we will see shortly). Consider the following example of a parametrized stack:

```
spec Stack[elem :: Type] = stack :: Type with
        new: stack
        push: stack * elem -> stack
        pop: stack -> stack
        top: stack -> elem
        isempty: stack -> bool
        -- constraints
```

Note that `Type` itself is a specification. Indeed, it is the most general specification in the sense that it does not specify any operations on its instance types.

Structures are instances of specifications and describe how abstract objects are implemented. The following structures give two alternative implementations of the specification `Stack`:

```
type liststack[e :: Type] :: Stack[e] = List[e] with
        new = nil
        push(l, e) = e :: l
        pop = tl
        top = hd
        isempty(l) = (l = nil)

type arraystack[e] = { a: Array[0..MAXSIZE] of e, i: 0..MAXSIZE } with
        new = { a = new[Array[0..MAXSIZE]], i = 0 }
        push(s, e) = s{a := s.a[i+1 := e], i := s.i + 1}
        pop(s, e) = s{i := s.i - 1}
```

```
        top(s) = s.a[s.i]
        isempty(s) = (s.i = 0)
```

They are functional implementations in the sense that they do not carry a state. A more "object-oriented," imperative implementation could be given by

```
refliststack[e] :: Stack[e] = ref List[e] with
        new = ref nil
        push(s, e) = (s := x :: !s; s)
        pop(s) = (s := tl(!s); s)
        top(s) = hd(!s)
        isempty(s) = (!s = nil)
```

Let us now demonstrate how specifications can be reused (inherited). From an abstract point of view, a queue is a stack with some more operations. When specifying a queue, it is natural to use the specification of `Stack` and tack on the additional operations.

```
Queue[elem:: Type] = Stack[elem] with stack as queue and
        append: elem * queue -> queue
        delete: queue -> queue
        last: queue -> elem
        isfull: queue -> bool
```

Note how the virtual representation `stack` is renamed to `queue` and used in the signature of the additional operations. The representation corresponds to "mytype" in typical object-oriented languages.

Having just seen reuse of specifications, let us introduce reuse of implementations, for which *G* provides a separate, quite flexible mechanism. Aiming at implementing queues, we start by extending the list implementation of a stack, and then giving an array-based implementation of a queue not inherited from any other structure.

```
listqueue[elem] :: Queue[elem] = liststack[elem] with
        (liststack[elem]): listqueue[elem] -> liststack[elem]
                        -- automatically inferred type conversion
        append(e, (c :: q)) = c :: append(e, q)
        append(e, nil) = [nil]
        delete(c :: nil) = nil
        delete(c :: l) = c :: delete(l)
        last(c :: nil) = c
        last(c :: l) = last(l)
        isfull(l) = false

arrayqueue[elem] :: Queue[elem] = { a: Array[0..MAXSIZE] of elem,
                i, j: 0..MAXSIZE, full: bool} with
        new = { a = new[Array[0..MAXSIZE], i = MAXNUM, j = 0, full = false }
        push(q, e) = if isfull(q) then raise full else
                q{a := q.a[j := e], j := next(q.j), full =
                        (prev(q.j) = q.i)}
        pop(q) = if isempty(q) then raise empty else
                q{j := prev(q.j)}
        top(q) = if isempty(q) then raise empty else
```

```
        q.a[prev(q.j)]
append(e, q) = if isfull(q) then raise full else
        q{a := q.a[i := e], i := prev(q.i), full =
                (prev(q.j) = q.i)}
delete(q) = if isempty(q) then raise empty else
        q{i := next(q.i)}
last(q) = if isempty(q) then raise empty else
        q.a[next(q.i)]
isempty(q) = not isfull(q) and (prev(q.j) = q.i)
isfull(q) = q.full
-- auxiliary definitions
next(i: int) = if i = MAXSIZE then 0 else i+1
prev(i: int) = if i = 0 then MAX
```

Having written such an implementation of a queue, one might be curious whether it can be made into a stack implementation simply by getting rid of part of the operations provided. By giving `arrayqueue` the signature `Stack`, we hide the operations in `arrayqueue` that are not part of `Stack`. As elaborated in section 2, existing object-oriented languages do not typically provide this flexibility:

```
arraystack2[elem] :: Stack[elem] = arrayqueue[elem]
```

3.3 Hidden Types and Dynamic Dispatching

Suppose we are given a specification of a type class `Point` along with some implementations (which we are not showing).

```
class Point = t :: Type with
        new: real * real -> t
        x: t -> real
        y: t -> real
        r: t -> real = sqrt(x(p)^2 + y(p)^2)
        theta: t -> real = arctan(y(p)/x(p))
        eq(p: t, q: t): bool = x(p) = x(q) and y(p) = y(q)
        translate(p: t, u: real, v: real): t =
                new(x(p)+u, y(p)+v)
        scale(p: t, s: real): t =
                new(x(p)*s, y(p)*s)
        sqdistance(p: t, q: t): real =
                (x(p) - x(q))^2 + (y(p) - y(q))^2
```

A generic point would then have the type

```
type AnyPoint = some P :: Point with |P|
```

Another example of dynamic dispatching, which also involves multiple inheritance, is given in Appendix A.

4 Type-theoretic Aspects

The *G* type system is based on an explicitly typed, predicative lambda calculus and can be considered as an extension of the type systems described in [Mac86,MH88,MMM91]. In this section, we will briefly review the concepts underlying those type systems and then outline the *G* type system.

XML is a language introduced in [MH88] in order to explain more precisely the type system of Standard ML, including its module facility. The type system of XML provides two universes of types, U_1 and U_2, which are informally called "small" and "large" types, respectively. This separation reflects the phase distinction between the static evaluation of modules and the dynamic evaluation of values in Standard ML. The universe of all values, U_0, contains all entities whose types are members of U_1. XML may be defined relative to an arbitrary collection of base types, e. g. integers and booleans, and user-defined algebraic free types. Its small types include any type expression constructed using only base types, monomorphic type variables, and the function space constructor $\rightarrow$.

The large type universe of XML corresponds to Standard ML's polymorphic types and module facility. U_2 contains U_1 itself, polymorphic functions, and types constructed from other members of U_2 using general sum and product operations. We will now review these operations as introduced in [Mac86].

Let A be a set, and B a family of sets indexed by A, meaning that $B(a)$ is again a set for each $a \in A$. Then the *general product* of A and B, written $\Pi x : A.B(x)$ is the set of functions f from A to the union $\cup_{x \in A} B(x)$ such that for each $a \in A$, we have $f(a) \in B(a)$, i. e.

$$\Pi x : A.B(x) = \{f \in A \rightarrow \cup_{x \in A} B(x) \mid \forall a \in A.f(a) \in B(a)\}$$

Note that in the degenerate case in which B is constant, the general product reduces to the function space $A \rightarrow B$.

The *general sum*, written $\Sigma x : A.B(x)$, for a set A and a family of sets B indexed by A, is the set of pairs $< a, b >$ such that $a \in A$ and $b \in B(a)$, i. e.

$$\Sigma x : A.B(x) = \{< a, b > \in A \times \cup_{x \in A} B(x) \mid b \in B(a)\}$$

In the degenerate case, the general sum reduces to the cartesian product $A \times B$. We may apply projection functions $|.|$ (witness) and *val* to members of general sum types which return the first and second component of the member, respectively.

Viewing types as sets, XML incorporates general sum and product types into U_2 by requiring that the index type A and the types contained in the family B be members of U_2. The following are examples of general sum and product types and members of such types:

$< int, 3 >$: $\quad \Sigma t : U_1.t$

nil: $\quad \Pi t : U_1.list(t)$

As seen in [Mac86,MH88], the signatures of Standard ML may be viewed as syntactic sugar for general sum types, and ML structures and functors as a notation for members of general sum and product types, respectively. Although functor signatures are not provided in ML, they could be described by general product types.

The language XML+ is presented in [MMM91] and can be seen as a generalization of XML incorporating several new features. It generalizes signatures by providing not only structure signatures

in form of general sums, but also functor signatures in form of general product types. In addition, XML+ features parametrized signatures, which have types of the form $U_2 \rightarrow U_2 \rightarrow \ldots \rightarrow U_2$, technically leaving the boundaries of U_2. Another feature included in XML+ is an impredicative treatment of existential types and a related implicit coercion from U_2 to U_1. We will give a brief overview of existential types and their use in XML+.

Existential types are introduced in [MP88,CW85] and model the programming language concept of type abstraction. Using the notation of [Mac86], an existential type is expressed as $\exists t : U_1.B(t)$, where B is a type expression possibly containing free occurrences of t. Values of such types are created by expressions of the form $\mathbf{hide}_{\exists t:U_1.B(t)}\tau M$, where τ is a U_1 type and M is an expression of type $B(t)$. The only expression available on members of existential types has the form **open** M **as** $x[t]$ **in** N. It has type ρ, assuming $M : \exists t : U_1.B(t)$ and $x : B(t) \Rightarrow N : \rho$, with the restriction that t does not leave the scope of N, i. e. appear free in ρ or the type of any variable appearing free in N. Locally within N, t refers to the type component, and x to the value component. Although existential types, similarly to general sum types, have a type component which is a member of U_1, they can be considered as members of U_1. This is possible because the type component is hidden and may be accessed only locally as an opaque type newly generated with each open operation.

XML+ provides signatures as general sum types; however, the structures described by those signatures may be treated as members of existential types so that they have small types. This flexibility is achieved by recognizing that there is a canonical coercion function

$$hide : \Sigma t : U_1.B(t) \rightarrow \exists t : U_1.B(t)$$

which simply hides the identity of the type component of a structure and is reflected in the typing rules.

G differs from XML in several ways. While maintaining a hierarchy of two type universes, it combines general sum types in U_2 with a modified notion of impredicative polymorphism and bounded existential types. Within the universe U_1, G identifies structures with the underlying representation types. We will now elaborate on these characteristics.

Polymorphism in G appears at two levels. Since it is desirable to treat polymorphic functions as first-class values, we consider polymorphic types such as $\Pi t : U_1.B(t)$, or commonly written $\forall t : U_1.B(t)$, as types in U_1. We can now write first-class functions parametrized by polymorphic functions, e. g.

$$g = \lambda f : (\forall t : U_1.t \rightarrow t).(f\ int\ 3, f\ bool\ false)$$

This option is described in [MH88], and the question whether the extension of XML with this sort of impredicative polymorphism is strongly normalizing still appears to be open. However, it is believed to be strongly normalizing [Mit90a], since the code of polymorphic functions is completely oblivious to the type parameter. Analogously, we view functors of the form $\forall s : U_2.B(s)$, where $B(s) : U_2$, as members of U_2, thereby allowing us to parametrize functors by other functors.

In addition to existential types such as $\exists t : U_1.B(t)$, G provides signature-bounded existential types of the form $\exists t : C.B(|t|)$ as U_1 types, where $C : \Sigma t : U_1.F(t)$ and $|t|$ denotes the first (witness) component of t (see [CW85,Mac86]). Such types are used to model partial abstraction, meaning that the only known property of the hidden structure is its membership in a signature C. They are useful when modeling object-oriented programming, where we have objects with a known interface but a hidden representation. As an example, consider

$$\begin{aligned} C &= \Sigma t : U_1.t \rightarrow \mathit{bool} \\ S &= < \mathit{int}, \lambda x : \mathit{int}.x \leq 0 >: C \\ s &= \mathbf{hide}_{\exists t:C.|t|} S\ 3 \end{aligned}$$

Now we can use the square operation when we locally open s as in

$$\mathbf{open}\ s\ \mathbf{as}\ x[t]\ \mathbf{in}\ (\mathit{val}(t))(x) : \mathit{bool}$$

In contrast to XML+, which models classes as general sum types and objects as their members, G has a 3-level hierarchy viewing objects as values, object types (classes) as U_1 types, and signatures (class interfaces) as U_2 types. At the level of U_1, we identify structures that group a small type with operations on that type with the witness type itself, which we view as a mere set of values. As seen in the previous example, C is a class interface specifying a square operation, S is a structure with interface C, and 3 is an object of class S, i. e. a value of type $|S|$.

G has a notion of signature conformity similar to XML's.

5 Conclusion

We have developed a language framework that integrates algebraic, functional, and object-oriented programming in a uniform way. We have shown that abstraction over structures plays a critical role in offering flexible manipulation of both homogeneous and heterogeneous data. This combines the advantages of algebraic and object-oriented programming.

Several important open problems remain. Type reuse and type derivation need to be worked out carefully to overcome the non-robust nature of literally copying structures. Furthermore, the generalized type inference problem of completing programs in our implicit language need to be addressed where types, structure modifiers and conversion functions may be elided.

Acknowledgments

We would like to thank all members of the Griffin group for extensive and lively discussions providing valuable stimulation and feedback; in particular, Malcolm Harrison for advocating a style of object-oriented programming based on algebraic data types rather than on pure objects; Chih-Hung Hsieh for proposing the use of typesets as a means of describing collections of types; and Edmond Schonberg for detailed comments on the paper. We would further like to thank Martin Odersky for helpful suggestions and for sharing his insights on locally polymorphic types with us.

References

BCGS89] V. Breazu, T. Coquand, C. Gunter, and A. Scedrov. Inheritance and explicit coercion. In *Proc. Logic in Computer Science (LICS)*, pages 112–129, 1989.

BHJL86] A. Black, N. Hutchinson, E. Jul, and H. Levy. Object structure in the Emerald system. In *Proc. ACM Symp. on Object-Oriented Programming: Systems, Languages, and Applications*, pages 78–86, October 1986.

CCH+89] P. Canning, W. Cook, W. Hill, W. Olthoff, and J. Mitchell. F-bounded polymorphism for object-oriented programming. In *Proc. Functional Programming and Computer Architecture*, pages 273–280, 1989.

CHC90] W. Cook, W. Hill, and P. Canning. Inheritance is not subtyping. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 125–135, Jan. 1990.

[CW85] L. Cardelli and P. Wegner. On understanding types, data abstraction and polymorphism. *ACM Computing Surveys*, 17(4):471–522, Dec. 1985.

[D+90] R. Dewar et al. Reference manual for the Griffin prototyping language. In progress. December 1990.

[EM85] H. Ehrig and B. Mahr. *Algebraic Specification*, volume 1/2 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.

[GR83] A. Goldberg and D. Robson. *Smalltalk-80: The Language and Its Implementation*. Addison-Wesley, 1983.

[Har86] R. Harper. Introduction to Standard ML (preliminary draft). Technical report, University of Edinburgh, February 1986.

[HW90] P. Hudak and P. Wadler. *Report on the Programming Language Haskell*, April 1990. (editors).

[Mac86] D. MacQueen. Using Dependent Types to Express Modular Structure. In *Proc. 13th ACM Symp. on Principles of Programming Languages*, pages 277–286. ACM, Jan. 1986.

[Mey88] B. Meyer. *Object-Oriented Software Construction*. Prentice-Hall, 1988.

[Mey89] B. Meyer. Static typing for eiffel. posted to comp.lang.eiffel, July 1989.

[MH88] J. Mitchell and R. Harper. The essence of ML. In *Proc. Symp. on Principles of Programming Languages*. ACM, Jan. 1988.

[Mit90a] J. Mitchell. Private communication, Nov. 1990.

[Mit90b] J. Mitchell. Toward a typed foundation for method specialization and inheritance. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 109–124, 1990.

[MMM91] J. Mitchell, S. Meldal, and N. Madhav. An extension of Standard ML modules with subtyping and inheritance. In *Proc. ACM Symp. on Principles of Programming Languages* Jan. 1991.

[MP88] J. Mitchell and G. Plotkin. Abstract types have existential types. *ACM Trans. on Programming Languages and Systems*, 10(3):470–502, 1988.

[Ode90] M. Odersky. Locally polymorphic types. submitted to SIGPLAN '91, Nov. 1990.

[OL91] M. Odersky and K. Läufer. Type classes are signatures of abstract types. Submitted to FPCA '91, March 1991.

[SDDS86] J. Schwartz, R. Dewar, E. Dubinsky, and E. Schonberg. *Programming with Sets: An Introduction to SETL*. Springer-Verlag, 1986.

[Sny87] A. Snyder. Inheritance and the development of encapsulated software components. In *Research Directions in Ojbect-Oriented Programming*, pages 165–188. MIT Press, 1987

[Str86] B. Stroustrop. *The C++ Programming Language*. Addison-Wesley, 1986.

[Uni83] United States Department of Defense. *Reference Manual for the ADA Programming Language*. Springer-Verlag, 1983.

[WB89] P. Wadler and S. Blott. How to make ad-hoc polymorphism less ad hoc. In *Proc. 16th Annual ACM Symp. on Principles of Programming Languages*, pages 60–76. ACM, Jan. 1989.

A A Collection of Examples in *G*

This section contains a number of examples that we considered during the design of our language.

A.1 Points, Circles, and Rectangles: Hidden Types and Dynamic Dispatching

We will start with another example of dynamic dispatching, which also involves multiple inheritance. It is based on the example in Section 3. Given the specification for `Point` from Section 3, we add two specifications, one for graphical objects and one for colored objects.

```
type Color = { red, green, blue }

spec ColorObj = colorobj :: Type with
        color: colorobj -> Color

spec GraphObj = graphobj :: Type with
        draw: graphobj -> void
        scale: graphobj * real -> graphobj
```

Using multiple inheritance, we obtain a new specification for graphical objects that also have a color:

```
spec ColGraphObj = ColorObj with colorobj as colgraphobj and
        GraphObj with graphobj as colgraphobj
```

Let us define some geometric objects we can draw. Note how they use the type AnyPoint we defined above, since we do not care how the points constituting the rectangle are represented.

```
spec CircleObj = GraphObj with graphobj as circle and
        new: AnyPoint * real -> circle
        center: circle -> AnyPoint
        radius: circle -> real
        scale: circle * real -> circle

spec RectangleObj = GraphObj with graphobj as rectangle and
        new: AnyPoint * AnyPoint -> rectangle
        lowerleft: rectangle -> AnyPoint
        upperright: rectangle -> AnyPoint
```

Here are implementations for Circle and Rectangle.

```
struct gencircle :: CircleObj = { center: AnyPoint, radius: real } with
        new(p, r) = { center = p, radius = r }
        center(c) = c.center
        radius(c) = c.radius
        scale(c, s) = c{ radius = c.radius * s }
        draw(c) = let <s, v> = c.center in
                  drawcircle(s.x(v), s.y(v), p.radius)

struct genrectangle :: RectangleObj =
                { lowerleft: AnyPoint, upperright: AnyPoint } with
        new(p, p') = { lowerleft = p, upperright = p' }
        lowerleft(r) = r.lowerleft
        upperright(r) = r.upperright
        scale(r, s) = let <t, v> = r.upperright in
                        r{ r.upperright = t.scale(v, s) }
        draw(r) = let <tl, vl> = r.lowerleft and
                      <tr, vr> = r.upperright in
                  drawrectangle(tl.x(vl), tl.y(vl), tr.x(vr), tr.y(vr))
```

Finally we come to the interesting part. We define a type for any graphical object and a function which locally dispatches such an object to its proper drawing function. Remember that such objects are actually pairs of a hidden type component, which gets bound to s, and a value of the representation type contained in the hidden type component, here bound to v.

```
type AnyGraphObj = some G: GraphObj with |G|

draw(g: AnyGraphObj) = let <s, v> = g in s.draw(v)
```

Now we can define a *heterogeneous* list type of graphical objects and draw such a list.

```
struct GraphList = List[AnyGraphObj] with
        draw: List[AnyGraphObj] -> void
        draw(nil) = ();
        draw(<s,v>::l) = s.draw(v); draw(l)
```

Hidden types actually offer more than what we just showed. The following example shows ho they can be used to group two points of the same type, where we only care that the type is th same, but not which particular implementation of the specification `Point`.

```
type APPair = some P :: Point with |P| * |P|

eq(pp: APPair): bool = |pp|.eq(pp.1, pp.2)
```

Using such a pair type, we can define a safe equality function that takes a pair of points an dispatches them to the equality function implemented in the common point structure P. Generall this makes binary operations on types dynamically dispatchable without endangering static typ safety or requiring multiple argument dispatching.

A.2 Based Set: Objects and Abstraction

Based sets play an important role in the implementation of the language SETL [SDDS86]. A specification level, a based set is parametrized by the element type and provides a set type and "translated" element type, both of which are abstract. Various set operations, overloaded for us with based or unbased sets, are provided.

```
spec basedSet[elem :: Eq] =
        type set
        type belem

        in: elem * set -> bool
        with: set * elem -> bool
        less: set * elem -> bool

        in: belem * set -> bool
        with: set * belem -> bool
        less: set * belem -> bool

        lookup: elem -> belem

        union: set * set -> set
        -- etc.
```

A simple, inefficient implementation of a based set can be given by representing the set as a bitvect and the translated elements as indices into the bitvector. In addition, we need a hidden variab representing the translation from indices to actual elements, implemented as a sequence[5] We gi implementations of the different set operations.

[5]The efficiency could be improved by using an associative data structure, e. g. a hash table.

```
struct basedSet[elem :: Eq] =
        type set = Seq[Bool]
        type belem = Nat
        type bset = Seq[elem]
        var base: bset

        in(b: belem, s: set): bool = if b > # base then false else s[belem]
        with(s: set, b: belem): set = s[b := true]
        less(s: set, b: belem): set = s[b := false]

        lookup(e: elem): belem = lookup(e, base)-- only this lookup is exported
        lookup(e: elem, bs: bset): belem =      -- this one is hidden
                if # s = 0 then
                  0
                else
                  1 + lookup(e, s[2..]) -- this is the inefficient part

        in(e: elem, s: set): bool =
               s[lookup(e)] handle subscript_error => false
        with(s: set, e: elem) =
                let l = lookup(e) in
                  if l = 0 then
                        base := base ++ [e]
                        s ++ [true]
                  else
                        s[# l := true]  -- assuming that this maintains the
                                        -- sequence contiguous by inserting
                                        -- appropriate false values
        less(s: set, e: elem) =
                s[lookup(e) := false] handle subscript_error => s

        -- union etc. implemented bitwise, base need not be updated
```

Note that the hidden variable `base` would be called a class variable in object-oriented terminology. Given this implementation, we can create variables of type `set`; the base is updated as operations on the variables are performed.

A.3 A Package for Vectors and Matrices: Multiple Implementations and Hidden Types

This example illustrates the use of a structure implementing several related types. We show a specification requiring two types, `vec` and `mat`, and operations involving these types.

```
spec VecMat[elem :: Num] =
        vec :: Type
        mat :: Type

        newvec: int * int -> vec                -- constructors
        newmat: int * int * int * int -> mat

        +: vec * vec -> vec                     -- arithmetic operations
        -: vec -> vec
```

```
-: vec * vec -> vec
*: vec * vec -> elem
*: mat * vec -> vec
*: vec * mat -> vec
*: mat * mat -> mat

[.]: vec * int -> elem                    -- indexing operations
[.]: mat * int -> vec
[,.]: mat * int -> vec
[.,.]: mat * int * int -> elem

-- etc.
```

Matrices and vectors may be dense or sparse. It is useful to provide implementations which efficiently cover both cases:

```
struct denseVecMat[elem] :: VecMat[elem] =
        type vec = Array[elem]
        type mat = Array[Array[elem]]
        -- etc.

struct sparseVecMat[elem] :: VecMat[elem] =
        type sparseVecElem = elem * int
        type sparseMatElem = elem * int * int
        type vec = Seq[sparseVecElem]
        type mat = Seq[sparseMatElem]
        -- etc.
```

We can now define generic object types for systems and solutions of linear equations. The type `anyLinEq` defines a tuple containing a vector and a matrix instantiated from the same, arbitrary implementation. We can code functions independently of the implementation of the objects passed as parameters; the Gaussian elimination function below works on dense or sparse vectors and matrices.

```
type anyLinEq = some vm :: VecMat with vm.mat * vm.vec
type anyLinSol = some vm :: VecMat with Seq[vm.vec]

gauss(leq: anyLinEq): anyLinSol =
        open leq as vm[vec,mat] in
                -- some Gauss elimintation operations
                -- using operations defined in VecMat
                -- computing seqOfSpanningVecs
                hide vm seqOfSpanningVecs
```

A Formal Specification for a Database Model with Object-Oriented Features

Regina Motz [1]
Décio Fonseca

Departamento de Informática
Universidade Federal de Pernambuco
PO Box 7851, 50739 Recife - PE - Brasil

ABSTRACT: In this paper we present the formal specification of an object-oriented database model, the E/D Model. The E/D Model is an extension of the E-R Model, where structural and behavioral properties are treated explicitly in an integrated way. The E/D Model also possesses features of object-orientation: objects, object identity, types, classes and multiple inheritance. Object models have an important part to play in the future of database systems, but there is no agreement about exactly what an object-oriented database is, in spite of several attempts to provide a definition. In order to avoid semantic problems with terminology, we believe that formal specification is a good way of present our model. Besides that, the formal specification allow us to rapid prototyping the model. The Zc notation is used and briefly explained in order to understand this paper.

1 Why Formal Specification ?

There are good reasons to give formal description of a database model:

- The problems of producing reliable software quickly and economically have been widely discussed in recent years, and initial experience has indicated that the use of mathematical methods of specifying and designing software can contribute towards a solution.
- It provides precise, implementation independent, description of concepts.

[1]Present address: Resistencia 1636/102, CP 11400, Montevideo - Uruguay
e-mail: pardo@incoiy.edu.ar

- A formal description may be regarded as a functional specification for the implementation.
- It provides a basis for reasoning about the *correctness* of programs -either directly, or by means of derived proof rules for correctness assertions.
- A formal description *documents* the design of the model.

On these approach of integration between Databases and Software Engineering the POESIS project [FS90] is been developed at the U.F.PE. The goal of the POESIS project is the construction of a multimedia environment for database modelling, implementation and access. The purpose is to provide a designer with an expressive tool for representing the information and to help a user to access this information. It provides a structured and cooperative menu-based interface [SFC90], an object-oriented management system (GOLGO) [MF89,Mac90,Sal89], and a knowledge base administrator. It is being developed a first data model, called E/D (an acronym for the words static/dynamic in Portuguese) [Ban89,Mot90], which is also being formally specified using the Zc notation. A prototype of the E/D Model has been implemented using Smalltalk/V. This prototype currently provides two data operations: schema defi nition and database creation.

2 An Overview of the E/D Database Model

This section introduces the basic concepts which the E/D Data Model deals with. The E/D Model is an extension of the E-R Model, where structural and behavioral properties are treated explicitly in an integrated way. The E/D Model also possesses features of object-oriented data models: objects (everything instance is an object) , object identity (objects have an unique identifier), types (same as abstract data type concept in programming languages), classes (agruoping objects) and multiple inheritance.

An *object* is a real-word element or concept which can be distinctly identified. The primary characteristic of an object is that it has an identity which persists through time, however the properties of the object may change. This identity may be considered to be represented by a system generated object identifier (oid). Objects may be explicitly created and destroyed.

Every object is an instance of one or more classes and every classe has an associated type. There is the usual notion of a type lattice, i.e. the possibility of multiple inheritance. The relation among the types is the common database relation of 'IsA' and the relation among the classes is the 'SubSetOf' relation (Figure 1).

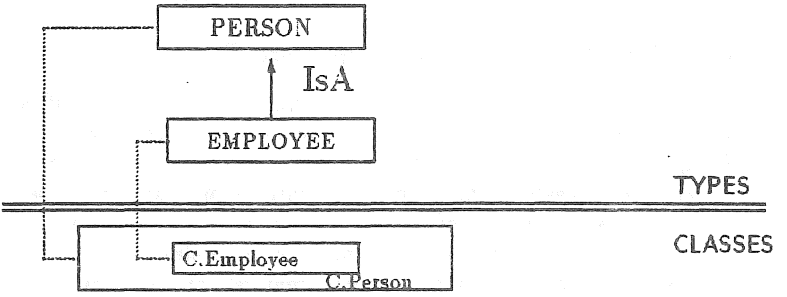


Figure 1: Relations of "IsA" and "SubsetOf".

A type can have multiple supertypes as graphically shown Figure 2.

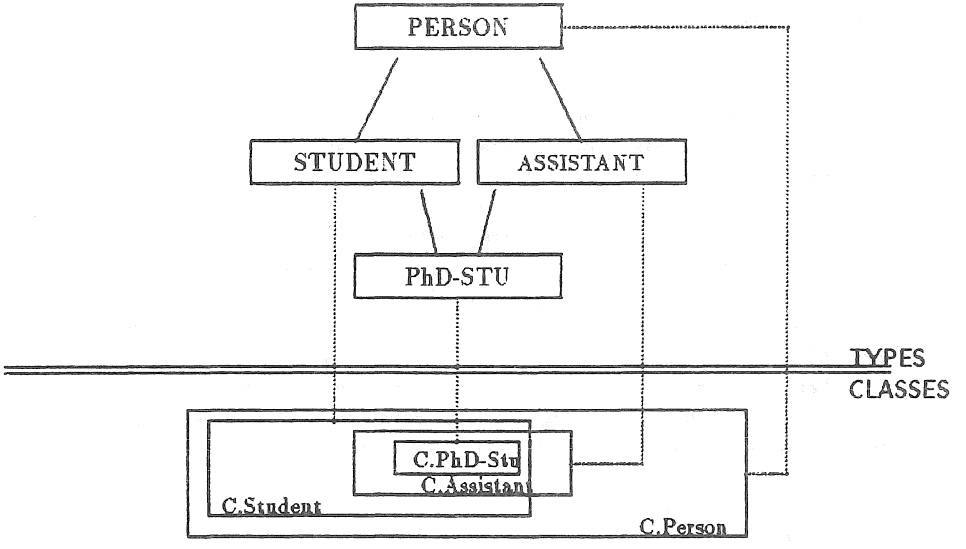


Figure 2: Multiple inheritance among types and classes.

The user must resolve conflicts between inherited structures or methods by writing a new structure or method, or explicitly inherit one instance variable or method from among several conflicting ones.

3 A Formal Description of the E/D Model

3.1 The Specification Language Zc

The Zc notation, a language for expressing mathematical specifications of complex computing systems, is been developed by th Software Engineering Group at UFPE [Sam88,Sou88]. It is based on the Z notation [Suf86], and as well as Z, is based on first order predicate calculus and typed set theory.

Zc increases the notion of *schema*, existing in Z, with the notions of *chapters*, *documents* and *library*, which improve its modularity and reusability.

A *schema* is a named piece of mathematical text that introduces some variables, and some predicates which express a constraint relating the possible values of the variables. Schemas are given in the form:

```
> SCHEMA Schema_S;
>    VAR
>       variables
> PREDICATE
>       predicates;
> END Schema_S;
```

Schemas can describe operations on data states. Some schemas require input and/or output data. Conventionally we decorate inputs with a final query, and outputs with a final shriek.

Although the style in which schemas are used in Zc is quite different from that used in Z, both languages have facilities for combining these structural units, and both make some provision for parametrized specification units.

In Z schemas are used for everything whereas in Zc there are specific syntatic units for defining functions (DEF), types (TYPE) and algebraic types (ALGTYPE).

A *chapter* clusters types, definitions, schemas and theorems, whereas a *document* groups reference to chapters and represents the whole specification of the system. Documents are given in the form:

```
> DOCUMENT DocName;
>       ChaptersNames;
> END DocName;
```

A *library* contains documents and chapters. Specifications of large systems are built up by specifying smaller sub-systems using chapters, then grouping these chapters into a document. Zc allows the importation and

exportation of objects[2] among the chapters and the definition of types and local definitions into a schema.

The documentation used is the one known as *literal programming*: lines beginning with the symbol > are lines of specification. The advantage of this type of documentation is that it allows to coexist in a single document the comments and the specification.

3.2 The Types of the E/D Model

In the E/D Data Model the type of an object may be system-defined or user-defined, and there is no disctintion between them. The system-defined types are: the atomic types (Integer, Real, String, Bool and Und[3]), the types constructs (Set, List and Record); and the predefined types of the E/D Model (UserT, TConstr, Entity, Rela, Action, Rule, Trans and MetaRule).

```
> ALGTYPE
>       Type :: user UserT          // User defined type
>             | restr TConstr       // Constraint
>             | ent Entity          // Entity
>             | rela Rela           // Relationship
>             | ac Action           // Action
>             | ru Rule             // Rule
>             | tran Trans          // Transaction
>             | metaru MetaRule;    // MetaRule
```

Types are organized in a type structure that supports generalization and specialization, this structure is a lattice. A type may be declared to be the subtype of another type.

Every type has a common information of it: a description, its supertypes, the excludents types with it and a set of operations.

```
> SCHEMA TypeInf;
>   VAR
>       description     : Message,
>       supertypes      : SET TypeName,
>       excludent       : SET TypeName,
>       operations      : Ops;
> END TypeInf;
```

[2] In this Section, the term *object* is used to reference to types, definitions, schemas and theorems.

[3] UND–The object undefined. This type is obtained every time an operation is not applicable.

The structure of a type is modelling as a parametrized record, it depends of the specific type. The representation of the structure as a record allows us to eficiently management the multiple inheritance.

```
> TYPE Structure[X] = MAP Label TO X;
```

3.2.1 Static Features

In this subsection we present the specification of the static features of the E/D Model: UserT, TConstr, Entity and Rela.

UserT

The user-defined types permit the extensibility of the model. Because the diversity of types required by the new applications, with multimedia documents, it is insufficient to extend the database system to include a specific and limited set of data types. For these reason the object-oriented approach allows the user to define methods on types in an incremental way. Our approch is similar in the sense that it lets the user define the data types and methods which are specific to the application area.

```
> SCHEMA UserT;
>   INCLUDE TypeInf.
>   VAR
>         struct      : Structure[TypeConstr],
>         constraints: SET NameTConstr;
> END UsuarT;
```

```
> ALGTYPE TypeConstr ::  ac AtomComp
>                        | set-of TypeConstr
>                        | seq-of TypeConstr
>                        | record Structure[TypeConstr];
```

```
> ALGTYPE AtomComp :: atomname Atom | typename TypeName;
>            Atom = Integer | String | Real | Bool;
```

TConstr

The type TConstr specify the constraints of the model. Exist four predefined constraints, which indicates that the values in the indicated labels of every object of a type with one of these constraints must be:

- defined for the *exist* one
- not null for the *not-null* one
- unique in the class for the *unique* one
- verify the condition for the *condition* one

```
> SCHEMA TConstr;
>    INCLUDE TypeInf.
>    VAR
>          struct: Structure[Constr]
> END TRestr;

> ALGTYPE Constr ::   exist (SET Label)
>                   | not-null (SET Label)
>                   | unique (SET Label)
>                   | dinamic Condition;
```

Entity

An entity concept has a name, a description, its parents, a structure, a set of restrictions and the set of total relationships in which the entity is a member. Moreover, each entity type in the database may posses at least one field in its structure then it has the following invariant: *the cardinality of the domain of its structure must be greater/equal than one.*

```
> SCHEMA Entity;
>    INCLUDE TypeInf.
>    VAR
>          struct      : Estrutura[AtomComp],
>          constraints: SET NameTConstr,
>          Totalrela   : SET RelaName;
> PREDICATE
>          CARD DOM struct >= 1;
> END Entity;
```

Relationship

Relationships have sets of entities associated with them, indicating the participating entities, as depicted in the E-R diagrams. Role functions allow to trace back the participating entities. Roles are connections between entities. E.g. the *ChildOf* relationship has the two roles *child* and *parent* that both must hold identifications to entities of the *person* type. Each role possesses an associated cardinality, which indicates the minimum and maximum times that an specific entity can participate in that role.

```
> TYPE Roles = MAP OpName TO Cardins;


> TYPE Cardins = (cardmin: INTEGER,
>                 cardmax: Cardin),
>       | (FORALL c: Cardins •
>       c.cardmin < c.cardmax AND c.cardmin >= 0);


> ALGTYPE Cardin = 'M' | int INTEGER;
```

It is possible to define constraints on the relationship's structure or on the participating entities, then exist the followinf type NameConstrRela in the specification:

```
> TYPE NameConstrRela ::   on-rela NameTConstr
>                        | on-partic (NameTConstr, Label);
```

A relationship concept has a name, a description, a set of entities partipating, a structure and a set of connections having the same role. It has also the following invariant: *each relationship in the database may be a relation between at least one entity type and there must be at least two roles.*

```
> SCHEMA Rela;
>   INCLUDE TypeInf.
>   VAR
>       partic      : SET NameEntity,
>       roles       : Roles ,
>       struct      : Structure[AtomComp],
>       total       : BOOL,
>       constraints : SET NameConstrRela,
> PREDICATE
        The relationship must have at least one participating entities
>       CARD partic >= 1 AND

        At least two roles must be defined
>       CARD DOM roles >= 2 AND

        The structure of the relationship includes the participating entities
>       partic ⊆ RNG struct AND

        The roles are defined among participating entities
>       FORALL opname:OpName | opname IN DOM roles •
>       Elems (operations opname).op-type ⊆ partic AND
```

```
        The constraints must be defined among existing labels in the
        relationship's structure or on the participating entities
>       FORALL c: NameConstrRela •
>       EXIST nc: NameTConstr, lab: Label | c IN constraints AND
>                 c = on-partic (nc, lab) •
>                 lab IN DOM struct AND (struct lab) IN partic;
> END Rela;
```

3.2.2 Dynamic Features

In this subsection we present the dynamic features of the E/D Model: Action, Rule, Transaction and Metarule.

Action

The action type allows the user to define its own conceptual operations over entities and relationships types. These conceptual operations use a set of primitives such as Insert, Delete, Modify, Let, and others pre-defined actions.

```
> SCHEMA Action;
>   INCLUDE TypeInf,
>   VAR
>       struct: Structure[ActionBody],
> END Action;

> TYPE ActionBody = (on : SET NameEntRela,
>                    act: SET OpActT);
```

Rule

The Rule type represents the behavioral properties of the entities and relationship types. It is a trigger in the model and can be seen as a conditional action.

```
> SCHEMA Rule;
>   INCLUDE TypeInf,
>   VAR
>       struct: Structure[RuleBody];
> END Rule;

> TYPE RuleBody = (action : NameAction, cond : Cond);
> ALGTYPE Cond :: cond-op CondOper | verif Condition;
```

```
> TYPE CondOper =(if : NameCondition, next-action : NameAction);
```

Transaction

The Transaction type groups actions and rules that modifies the state of one or more objects, which must be executed in a defined order and completed.

```
> SCHEMA Trans;
>    INCLUDE TypeInf,
>    VAR
>          struct: Structure[TransBody];
> END Trans;

> TYPE TransBody = (on  : SET NameEntRela,
>                   act : SEQ NameAcRule);
```

Metarule

The Meta-rule type allows the definition of priorities among the execution of several rules and controls the execution of exclusive ones.

```
> SCHEMA MetaRule;
>    VAR
>          struct: Structure[MetaRuleBody],
> END MetaRule;

> TYPE MetaRuleBody = (on             : SET NameEntRela,
>                      exclu-priority : SET (SEQ NameRule));
```

3.3 The E/D Instance and Scheme

The E/D Model consists of two part: the Instance and the Scheme.

Instance

An Instance of the E/D Model consists of a partial function (ObjMem) which associates an object for each object identifier.

```
> TYPE ObjMem = MAP Oid TO Object;
```

Each object has a state and an associated type (which is the most specialized type of it). These components of an object must verify an invariant which says that the structure of the state and the type must be in correspondence.

```
> TYPE Object = (sp-type: TypeConstr,
>                state:   ObjState);
>       | FORALL obj: Object •
>         (EXIST a: Atom | obj.sp-type = ac (atomname a) •
>         EXIST v: AtomicValue • obj.state = atomic v)
>       AND
>         (EXIST tc: TypeConstr | obj.sp-type = set-of tc •
>         EXIST s: SET Oid • obj.state = set s)
>       AND
>         (EXIST tc: TypeConstr | obj.sp-type = seq-of tc •
>         EXIST s: SEQ Oid • obj.state = sequence s)
>       AND
>         (EXIST n: TypeName | obj.sp-type = ac (typename n) •
>         EXIST m: MAP Label TO Oid •
>         obj.state = record m);
```

The structure of the object state (ObjState) shows that all the *type instances* has an uniformity representation as "objects".

```
> ALGTYPE ObjState = atomic AtomicValue |
>                   | set (SET Oid)
>                   | sequence (SEQ Oid)
>                   record (MAP Label TO Oid);

> ALGTYPE AtomicValue =  integer INT
>                      | real REAL
>                      | string STRING
>                      | bool BOOL
>                      | undef UNDEF;
```

Scheme

A Scheme contents the information about: Types and Classes.

```
> SCHEMA SchemeE/D;
> VAR
>     tenv    : TypeEnv,
>     classes : Classes;
> END SchemeE/D;
```

The type environment associates a type for each type name. It contains all the types defined in an application. Each type in the environment must verify that its supertypes or excludent types also exists in the environment, and that the excludent constraint is not violated.

```
> TYPE TypeEnv = MAP TypeName TO Type
>       | (FORALL Tenv: TypeEnv, t: Type |
>             t IN RNG Tenv •
>             t.supertypes ⊆ DOM Tenv AND
>             t.excludent ⊆ DOM Tenv AND
>             NOT EXIST st:TypeName | st IN t.supertypes •
>             (Tenv st).excludent ∩ t.supertypes <> {});
```

Each class has an associated type (which indicates the type of its objects) and an extension (which groups its objects).

```
> TYPE Classes = MAP ClassName TO ClassBody;
>       ClassBody = (type  : NameEntRelaUser;
>                    ext   : SET Oid);
```

DataBase

A DataBase consists of an instance (ObjMem) and a scheme, such that every object of the ObjMem belongs to a unique class of the scheme.

```
> SCHEMA DataBaseE/D;
> INCLUDE SchemeE/D
> VAR
>       objects: ObjMem;
> PREDICATE
>       FORALL id: Oid | id IN DOM objects •
>       UNIQUE c: ClassName | id IN (classes c).ext
> END DataBaseE/D;
```

3.4 Primitive Operations

The following are the primitive operations that are implemented in the current prototype of the E/D Model.

- Inicialization
- Create Type
- Generalization of a Type
- Insert operations in a Type
- Navegation among the Types
- Delete Type
- Create Class

- Delete Class
- Create Object
- Delete Object

At following we show the specification of the operation of create an user-defined type.

3.4.1 An Example: Create an user-defined type

```
> SCHEMA NewUserT;
>   INCLUDE NewInfType.
>   VAR
>         fields?   : SET Field[TypeConstr],
>         constr?   : SET NomeTConstr,
>         struct    : Structure[TypeConstr],
>         st        : Structure[TypeConstr].
>         t!        : UserT;
> PREDICATE
          The new type must be a valid type in the environment
>         PRED-ValidType AND

          The new type is generated
>         t!.description = message? AND
>         t!.supertypes = supertypes? AND
>         t!.excludents = excludents? AND
>         t!.struct = struct AND
>         t!.constraints = constr? AND
>         t!.operations = Make-Ops operations? AND

          The type environment is updated
>         Tenv' = Tenv $ {name? ↦ t!};
> END NewUserT;


> SCHEMA ValidType;
>   INCLUDE NewTypeInf,
>   VAR
>         fields?   : SET Field[TypeConstr],
>         constr?   : SET NameTConstr,
>         st1, st2  : Structure[TypeConstr]
> PREDICATE
          The fields of the structure of the new type
          are valid types in the environment.
>         VerifiySet (ValidField Tenv) fields? AND

          The structure of the type is built.
>         st1 = Make-Structure fields? AND
```

```
                The constraints must exists in the type environment.
>               constr? ⊆ DOM Tenv AND

                The structure of the type is a valid refinement of its supertypes.
>               VerifySet (Refine Tenv st1) supertypes? AND

                There is not clash among the multiples supertypes.
>               PRED-MultInherit AND

                The structure of the type with its supertypes is built.
>               st2 = BuildTypeStruct Tenv name? st1 AND

                The constraints are defined on existing labels of the type.
>               VerifySet (ValidConstr Tenv st2) restric?
> END ValidType;

> SCHEMA MultInherit;
>   INCLUDE SchmeE/D;
>   INCLUDE CommomInherit;
>   VAR
>           supertypes? : SET TypeName.
>           st?         : Structure[TypeConstr],
>           ssl         : SET(SET Label)
> PREDICATE
                Without multiples supertypes
>               CARD supertypes? =< 1 OR

                The set of labels of the type and its supertypes structure is built.
>               ssl = mapset (Labels Tenv) supertypes? AND

>               (CARD supertypes? > 1 ⇒
                Exists multiples supertypes without clash.
>             Clashes ssl = {} OR

                Exists clashes in multiples supertypes that may be resolved if the common label
                are redefined in the new structure of the type or
                if they belong to a common supertype.
>             Clashes ssl <> {} ⇒Clashes ssl ⊆ (DOM st? ∪ labs!))
> END MultInherit;
```

4 Conclusion

In our work, first of all we provided a brief review of the basic object-oriented concepts extracted from existing object-oriented systems in order to form the basis of our data model. We decided to emphasize on the notion of type. In database applications, the ability to add new data types allows one to model more easily and precisely a given application domain. User-defined domains may be described in the E/D Model, which makes

possible the inheritance of state variables and methods among domains, allowing the easy extensibility of the model.

We elaborated a formalization of our model using the specification language Zc. Zc was an adequate tool for the specification, especially by the fact that it provides a module structure. Chapter allows us to factor out common parts of a specification as can be seen in [Mot90]. This give us the opportunity to subordinate complexity. Formal specifications as the one shown in this paper can be a basis for direct prototyping. In our case, we obtained an object-oriented prototype in Smalltalk/V [Gol83] from the total specification of the E/D Model (shown in [Mot90]) using the method described in [MTM90]. With the help of this method the development of the prototype has been fairly easy and allowed in some sense the verification of the formal specification.

Last, but not least, we are exploring the integration of different programming languages with user-oriented caracteristics in order to help users to control their queries. The next step in order to achieve our objective is to develop a query language with object-oriented features in a formal way, i.e. give to it a formal semantics which will use this formal specification of the E/D Model as its semantic domains.

References

[Ban89] M. Bandeira. *Modelagem Estática/Dinâmica de Sistemas de Informações.* Master's thesis, Universidade Federal de Pernambuco, Recife, 1989.

[FS90] D. Fonseca and A. C. Salgado. Ambiente Integrado de Desenvolvimento de Banco de Dados Multimídia. In *Anais do XV Congresso Latino Americano de Informática*, CLEI, Asunción - Paraguay, 1990.

[Gol83] A. Goldberg. *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, 1983.

[Mac90] M. A. C. Macedo. *GOLGO: Gerenciador orientado a objetos.* Master's thesis, Universidade Federal de Pernambuco, Recife, 1990.

[MF89] M. A. C. Macedo and D. Fonseca. GOLGO: Gerenciador orientado a objetos. In *Anais do IV SBBD*, Simpósio Brasileiro de Banco de Dados, Campinas, 1989.

[Mot90] R. Motz. *Estudo Formal de um Modelo de Dados Orientado a Objetos.* Master's thesis, Universidade Federal de Pernambuco, Recife, 1990.

[MTM90] R. Motz, F. Tepedino, and S. Meira. From model based specifications to object-oriented prototypes. In *Anais do X Congresso da SBC*, Sociedade Brasileira de Computação, Vitória, 1990.

[Sal89] A. C. Salgado. O nível físico orientado a objetos de um SGBD multi-midia. In *Anais do IV SBBD*, Simpósio Brasileiro de Banco de Dados, Campinas, 1989.

[Sam88] A. Sampaio. *Zc: Uma Notação para Especificação de Sistemas Complexos*. Master's thesis, Universidade Federal de Pernambuco, Recife, 1988.

[SFC90] A. C. Salgado, D. Fonseca, and M. Casado. Umiformização de interfaces de comunicação em ambiente multimídia. In *Anais do V SBBD*. Simpósio Brasileiro de Banco de Dados. Rio de Janeiro, 1990.

[Sou88] R. Souto. *Especificação Formal de Software de Grande Porte - Um Exemplo Real*. Master's thesis, Universidade Federal de Pernambuco, Recife, 1988.

[Suf86] B. Sufrin. *The Z Handbook*. Programming Research Group, Oxford, 1986.

Creación dinámica de procesos en LINDA

Leonardo Balbiani
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria
Av. Intendente Cantilo s/n
Buenos Aires - Argentina

Mónica Bobrowski
Escuela Superior LatinoAmericana de Informática-ESLAI
CC 3193 (1900)
Buenos Aires - Argentina

Abstract

Procesamiento concurrente, paralelismo, procesamiento distribuido, son términos cada vez más difundidos. Una de las propuestas que aparentemente resulta ser bastante novedosa en este tema, es la de *Comunicación Generativa* y, más específicamente, Linda. El mecanismo de creación dinámica de procesos de Linda no ha sido considerado detalladamente en la bibliografía. El objetivo de este trabajo es analizar dicho mecanismo. Se presenta, en primera instancia, un breve análisis del modelo de *Comunicación Generativa* y de Linda en general, pasando luego a estudiar detalladamente la primitiva para creación de procesos del modelo. A continuación se propone una implementación para la misma tomando en cuenta las consideraciones planteadas al respecto. Se presentan algunos ejemplos sencillos de programas que utilizan dicha facilidad. Finalmente, se presentan las conclusiones del trabajo.

1 Introducción

Procesamiento concurrente, paralelismo, procesamiento distribuido, son términos cada vez más difundidos. Sin embargo, el estudio de estos campos en informática es relativamente reciente y abre un amplio espectro para la investigación y el desarrollo. Día a día se presentan nuevos modelos, nuevos lenguajes, nuevas herramientas que intentan mejorar, entre otras cosas, la expresividad, la elegancia, la sencillez y la eficiencia de las ya existentes. Una de las propuestas que, aparentemente, resulta ser bastante novedosa es la de *Comunicación Generativa* y, más específicamente, Linda. Esta idea fue desarrollada principalmente en la Universidad de Yale, en New Haven, Connecticut, U.S.A., por David Gelernter, Nicholas Carriero y otros.

El modelo de *Comunicación Generativa*, sobre el cual se basa Linda, ha sido ya ampliamente analizado. Lo mismo puede decirse con respecto a las primitivas para comunicación entre procesos de Linda. Sin embargo, el mecanismo de creación dinámica de procesos del modelo no ha recibido un tratamiento similar. Son escasas las referencias existentes en la bibliografía y más escasas aun cuando de implementaciones se trata.

El objetivo de este trabajo es, entonces, analizar dicho mecanismo. Para esto se realizó en primera instancia un breve análisis del modelo de *Comunicación Generativa* y de Linda en general, pasando luego a estudiar detalladamente la primitiva para creación de procesos del modelo. Se diseñó una implementación de esta primitiva sobre el kernel Linda descrito en [2].

La estructura del trabajo es la siguiente: En 2 se describe brevemente el modelo (para una descripción más detallada ver [8]), haciendo hincapié en ciertos problemas que surgen y comentando la implementación descrita en [2]. En 3 se analiza específicamente la primitiva para creación dinámica de procesos de Linda, sobre la cual la bibliografía existente es bastante escasa. Se muestran las ambigüedades de la semántica de esta primitiva y se comentan algunas decisiones tomadas al respecto. En 4 se estudia la implementación de esta primitiva sobre C extendido con el Kernel Linda propuesto en [2]. Se plantean los problemas de esta implementación y se analizan posibles soluciones. En 5 se presentan algunos ejemplos sencillos y bastante generales de programas que crean procesos dinámicamente y finalmente en 6 se presentan las conclusiones.

2 LINDA

Linda es un modelo de programación paralela basado en la idea de *Comunicación Generativa*, la cual unifica las nociones de creación de procesos, comunicación y sincronización. El concepto de *Espacio de Tuplas* es central a este modelo: una región de memoria (lógica) compartida asociativa que puede contener tanto objetos *pasivos* (datos), como procesos *activos* computándose en forma de *tuplas*.

Si bien el modelo forma parte de los conocidos como de *memoria compartida*, posee algunas particularidades dignas de mencionarse en este contexto:

- La unidad de almacenamiento en el espacio de tuplas es la *tupla lógica* -a diferencia de las memorias convencionales en las cuales la unidad de almacenamiento es, por ejemplo, el byte físico. Aquí los elementos son accedidos por *contenido*, i.e., cualquier selección de los valores de la tupla, para lo cual se necesita un mecanismo de *puttern-matching* -en las memorias standard, se accede simplemente por dirección.
- Las operaciones sobre el espacio de tuplas son *leer*, *insertar* y *remover* -sobre una memoria convencional son leer y escribir-, con lo cual las tuplas no pueden ser alteradas en el mismo espacio; deben primeramente ser removidas, luego alteradas y finalmente insertadas en el espacio de tuplas.

- La comunicación entre los procesos se da a través de la inserción y remoción (o lectura) de tuplas. Este tipo de comunicación es *desacoplada en tiempo* y *desacoplada en espacio* . Estas características son propias del modelo de memoria compartida. Se ve que la existencia de las tuplas es independiente de la de los procesos.

Linda consiste de unas pocas simples operaciones sobre el espacio de tuplas. Un lenguaje *base* con el agregado de estas operaciones puede transformarse en un lenguaje de programación paralela.

Las operaciones básicas de Linda sobre el espacio de tuplas son cuatro:

- **out**(t): La tupla t es agregada al espacio de tuplas. Los argumentos de t se evalúan *antes* de que la tupla sea insertada. Esta operación es *asincrónica*, i.e., el proceso que la invocó continúa su ejecución inmediatamente.
- **in**(s): Si existe alguna tupla t en el espacio de tuplas que haga matching exitoso con el *template* s, ésta es removida . Sus parámetros reales son asignados a los respectivos formales de s. Esta operación es bloqueante, i.e., si no existe matching exitoso, el proceso se suspende hasta que aparezca, en cuyo caso prosigue la ejecución del proceso invocante. Si el matching no es único, se elige una tupla de forma no determinística.
- **rd**(s): Esta operación es similar a **in**, sólo que la tupla elegida no es removida del espacio de tuplas.
- **eval**(t): es similar al **out**, pero difiere el momento en que los argumentos son evaluados. La tupla t se agrega al espacio de tuplas en forma *no evaluada*. Esta es la primitiva de creación de procesos de Linda que es objeto del presente trabajo, por lo tanto su descripción detallada se posterga.

Un *matching* entre una tupla y un template es exitoso si:

1. La tupla y el template tienen el mismo número de argumentos.
2. Argumentos correspondientes de la tupla y el template tienen el mismo tipo.
3. Parámetros reales correspondientes son iguales, donde la igualdad es la definida en el lenguaje base para objetos de ese tipo.
4. La tupla y el template no tienen parámetros formales correspondientes.

Nota: Algunas descripciones de Linda agregan como primitivas a **inp** y **rdp**, las cuales se comportan exactamente igual a **in** y **rd** salvo en lo que se refiere a bloqueo: **inp** y **rdp** nunca se bloquean; en lugar de ello devuelven un valor indicando éxito (si hallaron la tupla) o fracaso (en caso contrario). La implementación de estas primitivas se deriva fácilmente de las anteriores y no serán tenidas en cuenta en el presente trabajo. Para más detalles, ver [8].

El control del acceso a cada una de las tuplas del espacio de tuplas se distribuye entre los procesos, lo cual se conoce como *distributed sharing*.

En Linda (en principio) hay un solo espacio de nombres asociado a todos los procesos que conforman un programa. Éstos no tienen necesidad de indicar explícitamente cuál es el espacio de nombres asociado a cada uno. Esto se conoce como *distributed naming*. Según [14], todos los procesos Linda que son parte del mismo programa tienen acceso al mismo espacio de tuplas lógico. En algún sentido, es el acceso a este objeto compartido el que define qué procesos constituyen un programa Linda.

Con este esquema, algunos de los problemas que se presentan son:

- Protección: El espacio de tuplas no prevee la posibilidad de manejar problemas de *alias*, ni tampoco especificar una forma de impedir el acceso de un proceso a ciertas tuplas (el espacio de tuplas es *flat*).
- Duplicación: El espacio de tuplas es un *multiset*, i.e., puede haber tuplas repetidas. No existe un mecanismo para detectar que se está intentando insertar una tupla repetida. Sería deseable poder especificar que cierto conjunto de tuplas tiene la estructura de *set* (i.e., sin duplicaciones) y que sea el sistema el que garantice la consistencia.

Se han presentado propuestas que intentan dar solución a estos y otros problemas utilizando *multiples espacios de tuplas* ([10]). Este nuevo enfoque complica considerablemente el comportamiento de la primitiva de creación de procesos, modificando sustancialmente su semántica y generando nuevos problemas que no parecen tener una solución simple a nivel de implementación. En el presente trabajo se estudia la versión original del **eval**, que es un paso previo indispensable para recién tentar la comprensión de la nueva propuesta.

A continuación se describirá brevemente la implementación de un soporte *run time* de Linda con las primitivas **in**, **out** y **rd** sobre el cual se diseñó una implementación de **eval**. Para más detalles, ver [1], [2].

La 3B Computer Network sobre la cual se trabaja, está constituida por sólo dos nodos host -*elena* y *eslai*. Cada nodo es una minicomputadora 3B 2 con 2.8 MB de memoria central libre que corre UNIX System V extendido con facilidades de networking.

El espacio de tuplas se divide en dos subespacios, uno en cada nodo. Así, cada nodo se encarga de *administrar* su propio espacio debiendo satisfacer todos los pedidos de tuplas que pertenezcan al mismo. El criterio empleado para la división fue partición por función distribuida, i.e., una función que para cada tupla determina unívocamente el subespacio que le corresponde.

El subespacio de tuplas instalado en cada nodo está administrado por un proceso que *vive siempre*, independientemente de la aplicación, llamado *gestor*.

El gestor se encarga de asignar una tupla particular (generada por un **out**) a un *template* (generado por un **in** o **rd**).

Los requerimientos de las aplicaciones son codificados en mensajes y transmitidos a través de la red . Cada mensaje contiene la operación, la tupla y el identificador del proceso que lo envía. El nodo al que se dirige el mensaje es determinado por la función de partición del espacio.

Cada **in** o **rd** puede o no encontrar una tupla que haga matching. En el primer caso, el gestor contesta inmediatamente el mensaje recibido enviando la tupla encontrada. En el segundo, guarda el mensaje para esperar el arribo de la tupla correcta. El conjunto de mensajes bloqueados constituye el estado de la red o estado del sistema.

Cada **out** provoca que el gestor revise los mensajes retenidos en busca de templates que hagan matching con la tupla arribada. Si no hay ninguno, la tupla es instalada en el espacio local. En otro caso, se contestan los mensajes siguiendo alguna estrategia.

El nombre lógico de la tupla (primer campo que siempre es una constante de tipo string) juega dos roles en el sistema de run time: clave para determinar el nodo y clave para ordenar el espacio de búsqueda. La estructura de datos elegida para representar el espacio de tuplas asociado a cada nodo fue un esquema de hashing, usando el nombre lógico como clave. Además, es necesario mantener el estado del sistema -i.e. mensajes retenidos correspondientes a procesos bloqueados por operaciones **in** o **rd** para las que no hay match. Se decidió utilizar el mismo esquema de hashing usado para el espacio de tuplas.

La tabla de hash fue implementada como un arreglo de estructuras de dos campos. El primero es una lista de tuplas y el segundo una lista de mensajes.

La forma interna de una tupla es una estructura que contiene la longitud y un arreglo de componentes. Se impone un límite a la longitud de las tuplas para poder transmitirlas de un nodo a otro con un único paquete de datos. Cada componente es una estructura de 3 campos: el tipo de la componente, si es formal o real y su valor en este último caso.

Una vez construida la representación interna de la tupla, el siguiente paso consiste en enviarla al nodo correspondiente. Para ello se construye un mensaje con el nombre de la puerta de lectura del proceso A , la operación y la tupla. El mensaje es empaquetado en formato ETHERNET y enviado a través de la red. Después el proceso A se queda bloqueado sobre su puerta de lectura esperando que el gestor le conteste. Cuando el gestor recibe el paquete enviado por A, lo decodifica y traduce a un formato interno adecuado. Si en el subespacio correspondiente hay una tupla que hace matching, el gestor arma con ella un paquete (siguiendo el mismo camino descrito antes) y lo transmite por la red con destino a la puerta de lectura del proceso A. Cuando el paquete es recibido, se extrae la tupla y se asignan los valores reales a los correspondientes formales.

En el soporte de run time de C-Linda implementado, el proceso gestor del espacio de tuplas es instalado en el momento de *start-up* del sistema.

Esta implementación es sólo un prototipo. De todas maneras, adolece de ciertas fallas que no siempre están relacionadas directamente con la eficiencia:

- El soporte de comunicaciones no es confiable. i.e., no hay garantías de que los mensajes lleguen a destino.
- El espacio de tuplas es global a todos los programas. Esto puede generar problemas de *aliasing* entre tuplas de distintos programas. Esto implica que no se pueda garantizar el comportamiento de dos o más programas que compartan algún nombre de tuplas.
- Los procesos tienen que ser *cargados* directamente por el usuario en el nodo en el que serán ejecutados. No existe creación dinámica de procesos ni distribución de los mismos desde el programa de aplicación.
- Todos los mensajes van a parar a la red, incluso cuando el destino final de la tupla (o del template) es el mismo nodo en que fue generada.
- El programador está obligado a dar explícitamente la *signatura de tipos* (tipos de los argumentos) de la tupla. Esto es inevitable por tratarse de un soporte en tiempo de ejecución.

Estos problemas condicionan ciertas decisiones del diseño propuesto en 4.

Esto completa la descripción de Linda y del soporte sobre el cual se trabajará.

3 Creación dinámica de procesos en LINDA

A continuación se describirá la evolución de los mecanismos de creación dinámica de procesos en Linda a partir de la bibliografía existente, para luego llegar a la versión que será considerada final en este trabajo, la cual será analizada en detalle.

En [9] se trata la creación dinámica de procesos en Linda de la siguiente manera: "La creación dinámica de procesos es una capacidad de los operadores de comunicación generativa en sí mismos. Brevemente, cualquier expresión libre de efectos colaterales puede aparecer en una tupla generada, y las tuplas son agregadas al espacio de tuplas en forma no evaluada. Tan pronto como la tupla ingresa al espacio de tuplas, la evaluación de todas sus componentes no evaluadas comienza simultáneamente. Luego, la sentencia **out**(P, f(x)) genera un nuevo proceso; su nombre es "P" y ejecuta f(x). Si f(x) devuelve un entero, la tupla resultante puede ser extraída del espacio de tuplas por medio de la sentencia **in**(P, i:integer). Si **in**() se ejecuta mientras f(x) está siendo evaluada, **in**() se bloquea hasta que se completa la evaluación." Como puede verse, aquí no hay primitivas separadas para crear tuplas de datos y tuplas de procesos. Una tupla que tiene algún argumento no evaluado (por ejemplo, un llamado a una función) es insertada en el espacio de tuplas y da lugar a la creación de un proceso que va a evaluar ese argumento y que se va a ejecutar concurrentemente con el proceso que invocó **out**(). Si la intención es que los argumentos se evalúen *antes* de que la tupla sea insertada en el espacio de tuplas, la evaluación tiene que ser realizada *localmente* y luego armar la tupla poniendo el resultado de las respectivas evaluaciones como argumentos.

Posteriormente, en [11] se propone una nueva primitiva para creación dinámica de procesos, **eval**(t), con t tupla. La descripción de esta primitiva es la siguiente: " **eval**(t) es similar al **out**(t), salvo que **eval** agrega una tupla no evaluada al espacio de tuplas. La evaluación de t comienza tan pronto como la tupla ingresa al espacio de tuplas; sus campos son evaluados en orden arbitrario y, cuando se completa la evaluación, la tupla resultante puede ser leída usando **rd** o removida usando **in**, tal como cualquier otra tupla. El proceso que se crea implícitamente para realizar la evaluación tiene disponible para sí el valor de cada nombre que aparece en t así como ese nombre fue definido en el ambiente del proceso que ejecuta el **eval** (si este nombre es una función o procedimiento, su valor es una expresión ejecutable). Es un error invocar, dentro de una tupla **eval**, una función cuyo cuerpo hace referencia a variables no locales- sus valores no van a estar disponibles para el proceso que realiza la evaluación.

Por ejemplo: ejecutar

eval("Q", f(x))

provoca que la tupla

(**eval**("Q"), **eval**(f(x)))

sea agregada al espacio de tuplas. La evaluación de "Q" y la de f(x) proceden concurrentemente con el proceso que ejecutó el **eval**. Si f(x) evalúa a un entero j, la *tupla activa* (**eval**("Q"), **eval**(f(x))) se transforma en la *tupla pasiva* ("Q", j) cuando se completa la evaluación. Esta tupla puede ser removida con **in**("Q", int i) o leída con **rd**("Q", int i) como cualquier otra tupla." En el mismo artículo se describe un programa C-Linda: "La ejecución de un programa Linda (C-Linda) siempre comienza con la rutina llamada main() (como en los programas C); main() inicializa cosas, luego crea n *workers* haciendo n veces:

eval(worker())

y después termina." En esta versión se presentan varias novedades en cuanto a la creación dinámica de procesos en Linda que vale la pena comentar:

1. Se provee un mecanismo *explícito* para creación de procesos, diferenciándolo de la inserción de datos en el espacio de tuplas. De aquí en más, se puede entender que los argumentos de una tupla **out**, aún cuando no estén evaluados en el momento de la invocación, serán evaluados antes de armar la tupla que será insertada en el espacio de tuplas. Todos los argumentos de una tupla **eval** son evaluados después de que la tupla ha sido insertada en el espacio de tuplas, aunque en algunos casos esto parezca no tener mucho sentido (por ejemplo, si hay argumentos que son costantes). Esto mantiene la *homogeneidad* del modelo.

2. Se diferencian dos clases de tuplas: *activas* y *pasivas*. Las tuplas activas son argumentos de **eval** que están siendo evaluados. Las tuplas pasivas son generadas por **out** o son el resultado de la evaluación de una tupla **eval**. Aquí surge naturalmente una pregunta: una tupla activa, puede ser removida del espacio de tuplas? De ser así, qué significa remover o leer una tupla activa?

3 Se pueden ver dos niveles de concurrencia. Por un lado, varios **eval** pueden ser ejecutados concurrentemente (por ejemplo, los n workers del main se ejecutan concurrentemente). Por otro lado, los distintos argumentos de un mismo **eval** pueden ser evaluados concurrentemente.

4. Toda tupla activa degenera finalmente en una tupla pasiva. Esto es, todos los procesos creados tienen que producir finalmente algún resultado si terminan (los workers, por ejemplo, podrían ser procesos que no terminen). Si bien esto preserva la homogeneidad del modelo y tiene utilidad a efectos de *sincronización* entre procesos y en casos en que se necesita conocer un resultado, no siempre un proceso produce un valor, o no siempre el valor producido es utilizado. Esto podría conducir a llenar el espacio de tuplas con tuplas que no son necesarias. Un garbage collector tradicional no puede decidir qué tuplas eliminar. Se podría eliminar este requerimiento y permitir que sean los mismos procesos los que generen tuplas pasivas al terminar, si es necesario. Pero esto podría generar problemas de sincronización, además de aumentar el costo de las comunicaciones y restarle elegancia y homogeneidad al modelo.

5. El **eval** introduce la necesidad de chequeos que no son necesarios en el lenguaje host. Por ejemplo las variables a las que pueden acceder los argumentos de un **eval** no siguen las leyes de alcance del lenguaje soporte, por lo tanto se deben añadir nuevos chequeos estáticos al compilador. De aquí se desprende que no alcanza con tener un macro-expansor que traduzca una invocación **eval** en una secuencia de instrucciones en el lenguaje host.

Haciendo un análisis un poco más profundo de la semántica del **out** y la del **eval**, se puede concluir que, en realidad, a nivel del modelo la primitiva **out** no es necesaria. Todo lo que puede ser realizado por un **out** también puede ser realizado por un **eval**. La ejecución *local* asociada al **out** puede obtenerse ejecutando las funciones en el espacio local, asignando sus resultados a variables y luego poniendo estas variables como argumentos de la tupla a ser insertada por medio de **eval** en el espacio de tuplas. Los procesos creados para evaluar dichos argumentos simplemente devolverán el valor de los mismos. Es cierto que hay diferencias de eficiencia: con **out** no es necesario crear procesos para todo, pero la potencia es la misma.

Out de alguna manera funciona como una *macro* que, al expandirse, resulta en lo siguiente:

```
int f();

out(f(x))
```

se traduce en:

int a;

a:= f(x);

eval(a): insertar la tupla (a) en el espacio de tuplas

Con el **out** propuesto originalmente en [9], que hace la distinción entre los argumentos para los cuales es necesario crear procesos que los evalúen y aquéllos para los cuales no, ni siquiera existen diferencias de eficiencia. Con esta sola primitiva se cubrirían tanto la creación dinámica de procesos como la inserción de tuplas de datos y por ende la comunicación entre procesos. De todas maneras, se ve que la implementación de la idea original no parece ser muy viable: se necesitaría poder reconocer estáticamente qué argumentos necesitan de la creación de un proceso para ser evaluados y cuáles no, lo cual puede no ser sencillo de detectar en tiempo de compilación para todas las tuplas que puedan ser generadas en un programa.

La información sobre **eval** no abunda en la bibliografía. Más escasa aún es la información sobre implementaciones de **eval**. Por ejemplo, en [8], donde se comentan alternativas para el diseño de un compilador para Linda, no se menciona el **eval**. Una de las pocas fuentes de información al respecto es [4]. Allí se comenta:

"**eval**(t) Es igual al **out**(t), excepto que t es evaluada después en lugar de antes de ingresar al espacio de tuplas; **eval** implícitamente hace *fork* de un nuevo proceso para realizar la evaluación.

La operacion **eval** es el mecanismo de creación de procesos del modelo de programación Linda (los sistemas Linda de memoria compartida emplean fork para crear los nuevos procesos especificados por un **eval**. Estos procesos son administrados por el sistema operativo multiprocesador y se manejan para ser ejecutados en procesadores libres). La implementación TSnet del **eval** provee ejecución remota de procesos a través del uso de las operaciones **in** y **out** de Linda.

Cada vez que se ejecuta una operación **eval**, una descripción del **eval** se inserta en el espacio de tuplas. La descripción incluye identificadores para campos de función y listas de argumentos. Todos los nodos, salvo el nodo maestro, continuamente corren un loop **eval** que hace **in** de alguna descripción disponible de **eval** y ejecuta las funciones especificadas. Una vez que una evaluación particular se completó, la tupla resultante se inserta en el espacio de tuplas y se obtiene una nueva descripción de **eval**."

Esta implementación es susceptible de ser criticada por varias razones:

- No se comenta cómo hacen los distintos nodos para conocer las funciones que pueden llegar a tener que ejecutar. Este punto es uno de los más dificultosos a la hora de la implementación .
- Desde el momento en que es *levantada* y hasta que se inserta la tupla pasiva correspondiente en el espacio de tuplas, la tupla parece no existir como tal.

- La descripción del **eval** involucra la inserción por parte del sistema de una tupla de nombre lógico "eval" como primera componente. Esto provoca problemas si un usuario inserta o remueve una tupla con ese nombre lógico. Contando solamente con un soporte en tiempo de ejecución esta dificultad no puede ser subsanada. Un compilador podría resolver el problema restringiendo el uso de tuplas con primera componente "eval" por parte del usuario.

- El proceso que *levanta* tuplas **eval**, como usa la primitiva **in** que remueve tuplas por el pattern-matching descrito anteriormente, si no se quiere que dependa del tipo de los argumentos que puede tener la tupla (i.e., que no haya un proceso distinto por cada posible signatura de tipos que pueda haber), debería manejar internamente todos los argumentos como pertenecientes a algún tipo unificado, con lo cual se necesita codificar de alguna manera la tupla como ese tipo antes de insertarla en el espacio de tuplas.

En [7] se describe una implementación de **eval** similar a la de [4]. La única aclaración que agrega es que los descriptores de **eval** contienen punteros a procedimientos, cuyas imágenes están cargadas en todos los nodos e incluyen también los valores de todas las variables mencionadas explícitamente en la invocación a **eval**. No se aclara cómo se obtiene la dirección de los procedimientos en cada nodo ni tampoco cómo y cuándo son cargados en ellos.

En [12] se dice que cuando se encuentra un **eval** se generan, para su evaluación, tantos procesos como argumentos tiene la tupla hallada. También se aclara que la tupla va a ser activa por un tiempo, pero tan pronto como sus argumentos han sido evaluados, la tupla que queda es una tupla pasiva como cualquier otra.

Vale la pena destacar que muchos de los problemas del **eval** no tienen que ver con una implementación particular de Linda, sino con la definición misma del mecanismo de creación dinámica de procesos. Muchos detalles que hacen a su semántica quedaron librados al azar, provocando serios problemas no sólo al intentar una implementación, sino también al intentar programar usando **eval**. Entre otras cosas, no queda claro:

- En qué momento se realiza el *binding* de las variables de las funciones que son argumentos de un **eval**.
- Cuál es el ambiente en que se ejecutan los argumentos de un **eval**.
- Qué cosas puede hacer una función que es argumento de un **eval**.
- Qué clase de parámetros son permitidos en una función que es argumento de un **eval**.
- Qué valores pueden ser resultado de la ejecución de una función que es argumento de un **eval**.

Para diseñar una implementación de **eval** se debe tomar decisiones sobre la interpretación de los puntos anteriormente citados.

4 Alternativas de diseño de EVAL

En esta sección se analizará detalladamente una implementación de la primitiva **eval** sobre el kernel Linda ya descrito en 2. En primer lugar se comentarán cuáles son los aspectos a tener en cuenta al diseñar una tal implementación, los problemas que surgen, las modificaciones y restricciones que deben imponerse al lenguaje *host* (de acuerdo a lo comentado en 3), para luego pasar a explicar cómo se resolvieron dichos inconvenientes en este caso. Cabe destacar que quedan varias cosas en el tintero y que, evidentemente, muchas de las decisiones de diseño tomadas pueden ser mejoradas.

En primer lugar, hay que considerar el equipo sobre el cual se va a trabajar: arquitecturas MIMD con memoria compartida o no, redes de computadoras, etc. Esto, si bien incide en la implementación de un kernel Linda en general, cobra mayor importancia cuando se trata de creación dinámica de procesos, ya que puede condicionar algunas de las decisiones a tomar. En una máquina de memoria compartida, el espacio de tuplas se instala en un segmento de memoria compartido por todos los procesos. Muchos de los problemas que se comentan luego y que están relacionados con la implementación de Linda sobre una red de computadoras no aparecen en ese tipo de máquinas. La principal dificultad reside en que en una red cada nodo tiene su propio espacio de memoria, sus propios periféricos y su propio sistema operativo que puede diferir del de otro nodo. Es importante recordar que la evaluación de un **eval** puede hacerse en un nodo distinto de aquel en el cual fue invocado y que dicho nodo no es conocido por el llamador.

Entre los aspectos importantes a considerar se cuentan:

- Restricciones al lenguaje host: no todas las construcciones del lenguaje host pueden ser usadas en las funciones que son argumentos de un **eval** o como argumentos del mismo (aun si no se consideran las expresiones que tienen efectos colaterales y se prohibe el uso de variables globales en las funciones que son argumentos de un **eval**). Incluso se puede llegar a tener que restringir ciertas cosas en general por no tener sentido semánticamente o por provocar problemas que no permitan garantizar cuál será su comportamiento. Algunos ejemplos que se pueden dar en distintos lenguajes son:
  - Archivos: pueden ser usados?, bajo qué condiciones?, pueden ser argumentos de tuplas?. El uso de archivos dentro de las funciones puede no tener sentido, ya que la única forma de compartir datos que podría permitirse en C-Linda es a través del espacio de tuplas.
  - Variables del tipo static: Su uso puede no tener sentido si las funciones pueden ejecutarse alternativamente en diferentes nodos. Lo mismo sucede con las cosas declaradas como *extern* (en Fortran, por ejemplo, habría que analizar qué sucede con las áreas declaradas *common*).
  - Punteros: Los punteros hacen referencia a una locación de memoria dentro de un determinado nodo. Qué sentido tiene transferir un puntero o hacer referencia a él si no se conserva también información sobre el nodo donde su valor está definido.

Aun el contar con esta información no sería suficiente, ya que se necesitaría contar con algún mecanismo de sincronización de acceso al valor apuntado.

- Fork: Se permite utilizar otro mecanismo de creación dinámica de procesos fuera de **eval**? Tendría sentido darle semántica a, por ejemplo, **eval**(fork())?
- Llamadas al sistema en general: Cuáles son las consecuencias de no conocer el nodo donde serán invocadas ciertas llamadas al sistema?
- Definiciones de tipos: Cómo son conocidas por todas las funciones que pueden ser ejecutadas en nodos no determinados de la red?
- Etc.

Una de las características de las primitivas de Linda más destacada en la bibliografía, es la *ortogonalidad* de las mismas con respecto a cualquier lenguaje *host* dentro del cual pueden ser implementadas. Esto parece ser consecuencia de la falta de consideración que ha recibido el **eval** frente al resto de las primitivas del modelo. Como se ve, Linda no parece ser *ortogonal* al lenguaje host. Los mayores problemas surgen al intentar implementar **eval**.

- **Eval** remoto vs. **eval** local: Se podría implementar un **eval** local, haciendo que las tuplas se evalúen en el mismo nodo en que son generadas, concurrentemente con el proceso que las genera. Por otro lado, se puede pensar en una implementación que distribuya las tuplas a ser evaluadas a lo largo de la red, de acuerdo a diversos criterios.

- Balance de carga: Al implementar un **eval** remoto, puede considerarse el balance de carga del sistema a efectos de determinar la distribución de las tuplas. Se puede considerar (dependiendo de las facilidades provistas por el sistema) el balance de carga *dinámico*, esto es, en tiempo de ejecución tomando en cuenta *todos* los procesos en ejecución en ese instante, o el balance de carga *estático*, esto es, determinar el destino de una tupla activa en tiempo de compilación de acuerdo *sólo* a la carga introducida en el sistema por el programa. También podrían considerarse híbridos.

- Nivel de distribución de los procesos: La distribución de los procesos puede darse a nivel de **eval** (i.e., distintos **eval** se evalúan concurrentemente) o a nivel de argumentos de un mismo **eval** (i.e., los distintos argumentos de un mismo **eval** se evalúan concurrentemente).

- Pasaje de parámetros a los argumentos de un **eval**: Si se implementa un eval remoto, no tiene sentido hablar de pasaje de parámetros por referencia a funciones que son argumentos de un **eval**. En el caso de C, esto se traduce en no poder usar punteros como argumentos de dichas funciones ni tampoco devolver punteros (C tiene sólo pasaje de parámetros por valor). El alcance de esta restricción es mayor, ya que en C los tipos estructurados sólo pueden pasarse por medio de punteros. Se ve que ésta es una seria limitación de **eval**. Notar que el problema aquí no es el hecho de pasar punteros en sí -se podrían pasar los objetos apuntados por ellos- sino cómo chequear que estos valores no han sido modificados, para evitar inconsistencias.

- Cómo se conocen las funciones: Un punto delicado es cómo conocen todos los nodos las funciones que eventualmente pueden llegar a ejecutarse en ellos. En otros lenguajes y mecanismos para procesamiento distribuido (por ejemplo, [6]) las funciones *residen* en determinados nodos y se puede manejar una tabla que diga, para cada nombre de función, dónde se la puede encontrar. Si la información del sistema no está centralizada, se puede guardar una copia de esta tabla en cada nodo y actulizarla mediante algún mecanismo de *broadcast*. En el caso de Linda, de alguna manera las funciones *viajan* junto con las tuplas. No siempre es posible determinar con antelación dónde se va a ejecutar una determinada función. Incluso se puede llegar a ejecutar en diferentes nodos simultáneamente. Este tema es bastante complejo, ya que en alguna medida la eficiencia de la implementación puede depender de él. Una solución sería cargar *a mano* una tabla con todas las funciones que pueden ser llamadas como argumento de eval en todos los nodos de la red. Se ve que esta solución no es la óptima. Más adelante se propone otra alternativa que presupone la existencia de un preprocesador.

El diseño que se detalla a continuación se basa en la implementación de un kernel Linda sobre C que se describe en [3], [1], [2] y que ya fue comentado en 2.

Los tipos permitidos como argumentos de tuplas pasivas son entero, real y string (constantes o variables). El tipo string, que en C es un puntero a caracter, se permite, a pesar de lo comentado anteriormente con respecto a punteros, ya que se asume como constante cuyo valor no se modifica, aunque no se chequea que esto realmente ocurra.

Los tipos permitidos como argumentos de tuplas activas son entero, real, string (constantes o variables) y funciones que reciban argumentos de esos tipos y que devuelvan valores de esos tipos. Con respecto al tipo string, vale el comentario hecho en el caso de tuplas pasivas.

Las tuplas activas y las tuplas pasivas se manejan por separado, esto es, se manejan realmente dos espacios de tuplas. Este punto es transparente al programador. Esta decisión se tomó para facilitar el manejo de las tuplas activas y para lograr un mayor nivel de protección (por ejemplo, evitar que se intente remover o leer tuplas activas). Existe un único gestor para ambos espacios de tuplas, cuyo funcionamiento se describe más adelante. La idea es aprovechar la implementación existente, minimizando las modificaciones. Para ello, se exige que las tuplas activas tengan un *nombre*,esto es, una constante de tipo string que las identifique. Esto permite el posterior armado de la tupla pasiva a insertar y su inclusión en el espacio de tuplas usando la primitiva out ya implementada.

Se debe contar también con la *signatura de tipos* de la tupla activa (ver [3], [1], [2]). En un primer momento, se puede exigir al programador incluirla como uno de los campos de la tupla, para luego contar con un preprocesador que se ocupe de generarla. En el caso de las tuplas activas existen dos alternativas (notar que el problema se presenta porque se deben incluir nombres de funciones y los argumentos con los que serán llamadas): una es incluir en la descripción de una función el número de argumentos además del tipo de la función y tomar los siguientes n campos de la tupla armada (donde n es el número de argumentos de

la función) como argumentos. Otra opción es identificar de manera especial los campos que son argumentos de una función. Vale la pena destacar que este problema no se presenta en el caso de las tuplas pasivas, ya que en éstas lo argumentos se evalúan *antes* de armar la tupla que será insertada.

Por ejemplo, la signatura de tipos de la tupla

eval("tupla", f(x), w, g(y))

(con f, g funciones que toman un entero y devuelven un entero; w,x,y enteros) sería, según la primera alternativa:

"sfi1vivifi1vi"

y según la segunda alternativa:

"sfiavivifiavi"

donde:

- s: constante string
- f: nombre de función
- i: entero
- v: variable
- a:argumento

Notar que, como los argumentos de las funciones de una tupla **eval** sólo pueden ser pasados por valor, en realidad, en el caso de argumentos, no es necesario distinguir si originalmente eran variables o constantes.

A su vez, el espacio de tuplas activas en cada nodo se divide en dos partes: la tabla de tuplas activas en espera (que no han sido *levantadas*) y la tabla de tuplas ejecutándose. La idea de esto es que una tupla activa no puede ser removida del espacio de tuplas activas hasta no haber derivado en una tupla pasiva. O sea, mientras una tupla está siendo evaluada, es necesario tener alguna forma de saber que existe. Si una tupla activa es *levantada* del espacio de tuplas y recién insertada la tupla pasiva correspondiente al finalizar la evaluación, puede haber problemas de consistencia. Mantener información sobre las tuplas que están siendo evaluadas permite eventualmente implementar mecanismos de recuperación de fallas y extender el soporte para considerar, por ejemplo, lectura y remoción de tuplas activas. Notar que la extracción final de la tupla activa del espacio de tuplas activas y la inserción

de la tupla pasiva correspondiente en el espacio de tuplas pasivas debería ser una operación atómica. Sin embargo, si bien esto es cierto a nivel conceptual, no es realmente necesario en la implementación, ya que no se permite la lectura o remoción de tuplas activas y la primitiva in es bloqueante.

La tabla de tuplas activas en espera contiene una entrada por cada tupla activa que ha llegado a ese nodo y que no ha comenzado su evaluación. La tabla de tuplas ejecutándose tiene una entrada por cada tupla que ha comenzado a ser evaluada y guarda sólo la signatura de tipos de la tupla -eventuales extensiones podrían exigir más información. Ambas tablas se guardan en un segmento de *memoria compartida* provisto por Unix, lo que permite que sean compartidas por un padre y sus hijos. La sincronización del acceso a las tablas se da por medio de *semáforos* también provistos por el sistema. Cada tabla tiene un semáforo asociado para garantizar que se acceda a ellas bajo *exclusión mutua*.

En cada nodo existirá un proceso que se encargará de *levantar* tuplas activas y mandarlas a ejecutar. Este proceso (para evitar hacer *busy-waiting*) será activado por medio de un semáforo asociado a él. El gestor del espacio de tuplas será el encargado de despertarlo al arribar una tupla activa, si es que no está despierto. El proceso levantará una tupla activa de la tabla, la mandará a ejecutar y luego buscará más tuplas activas. En caso de hallarse vacía la tabla, se dormirá por medio de su semáforo asociado. En la inicialización del kernel Linda debe incluirse la inicialización de los semáforos a utilizar, además de la creación de tablas, gestores, procesos, etc.

A continuación se describirán, con mayor detalle, los procesos que se añaden al núcleo Linda ya existente y se explicará su funcionamiento.

El gestor de operaciones del nodo origen debe armar la tupla activa a ser enviada, determinar a qué nodo de la red debe *viajar* esta tupla y finalmente enviarla.

Una tupla activa que *viaja* a través de la red contiene:

- la signatura de tipos de la tupla (que puede ser parte de la tupla original o ser armada por el preprocesador).
- la información de los argumentos de la tupla (funciones, argumentos, etc.).
- número de eval asociado (permite obtener, como se verá luego, la información necesaria para evaluar los argumentos de la tupla).
- nodo donde fue compilado el programa del que forma parte (información que permite optimizar el pedido de funciones para ejecutar si no están residentes en el nodo donde se llevará a cabo la evaluación, lo cual hace que no sea necesario guardar todas las funciones en todos los nodos de la red).
- señal de tupla activa (para saber en qué espacio de tuplas debe guardarse).

Para distribuir las tuplas activas entre los nodos de la red, se usa, en esta etapa de la implementación, una función aleatoria que determina un nodo entre los componentes de la

red. Notar que no se utiliza el mismo sistema que para las tuplas pasivas, o sea, hash sobre el *nombre* de la tupla, si bien de todas maneras se exige que las tuplas tengan nombre, como se explicó anteriormente. Lo ideal sería que la elección del nodo se realice en función de otras consideraciones, como por ejemplo, balance de carga dinámico de la red (determinando, en tiempo de ejecución, la carga del sistema), balance de carga estático de la red (determinando, en tiempo de compilación, la carga que va a imponer el programa al sistema y asignando en ese momento nodos a las tuplas activas), etc. Esto tiene que ver con que la distribución de procesos se da a nivel de *tuplas* y no a nivel de *argumentos* de las tuplas (en cuyo caso habría que hacer otro tipo de consideraciones), o sea, se crea un proceso, en algún nodo, para evaluar una tupla eval completa.

El gestor del espacio de tuplas de un nodo debe ser:

```
Gestor()
{
Leer de la red
Si es tupla pasiva
  entonces como antes
si no
  P(sem-tabla-tuplas-activas)
  poner tupla en tabla de tuplas activas
  V(sem-tabla-tuplas-activas)
  Si No-despierto(levanta-eval)
     entonces V(sem-levanta-eval)
}
```

donde P y V son las operaciones usuales sobre semáforos.

El proceso *levanta-eval* debe ser:

```
Levanta-eval()
{
loop
  P(sem-tabla-tuplas-activas)
  Si tabla vacia
    entonces { V(sem-tabla-tuplas-activas)
               P(sem-levanta-eval) }
  Sacar tupla activa
  V(sem-tabla-tuplas-activas)
  Pedir informacion para ejecucion
  P(sem-tabla-tuplas-ejec)
  Poner tupla en tabla de tuplas ejec
  V(sem-tabla-tuplas-ejec)
  Ejecutar tupla
```

```
forever
}
```

Pedir información para ejecución es testear si el archivo que será ejecutado para evaluar la tupla -el cual se determina a partir del número asociado al eval que contiene la tupla activa y que es generado por el preprocesador- se encuentra en el nodo actual. De no ser así, se envía un mensaje al nodo donde fue compilado el programa (información que se encuentra en la tupla activa), para que mande una *copia* de ese archivo. El archivo no se transfiere porque se pueden necesitar varias instancias del mismo al mismo tiempo en diferentes nodos.

Ejecutar tupla va a crear un proceso hijo que se va a encargar de evaluar los argumentos. Entonces, se tiene:

```
Ejecutar tupla()
{
Fork()
si es el padre
   entonces mandar argumentos al hijo
si no, ejecutar archivo asociado.
}
```

El *archivo asociado* es creado por el preprocesador. Se crea un archivo para cada tupla eval cuyo nombre queda unívocamente determinado a partir del número asociado al eval, que también es generado por el preprocesador. Este archivo contiene el código de las funciones que son argumentos de dicho eval y de todas las que pueden ser invocadas por ellas. Recibe los argumentos de la tupla y genera los llamados a funciones necesarios para poder evaluarlos. Tiene variables declaradas para almacenar los resultados de la evaluación de los argumentos y armar posteriormente la tupla pasiva. Tiene punteros declarados a todos los tipos de argumentos que pueden recibir las funciones, con lo cual la cantidad puede ser determinada en tiempo de ejecución. Tiene acceso a las primitivas Linda. Finalmente, arma la tupla pasiva a insertar, remueve la entrada correpondiente en la tabla de tuplas ejecutándose e inserta la tupla pasiva en el espacio de tuplas, por medio de la primitiva out.

La implementación de eval descrita es un prototipo y claramente puede ser mejorada. La simplicidad guió muchas de las decisiones tomadas. Sin embargo, a pesar de no buscar eficiencia se ve la necesidad de imponer restricciones al lenguaje host, en cuanto al tipo de argumentos que puede tener un eval y en cuanto al tipo de argumentos de los argumentos de un eval. Por ejemplo, los parámetros sólo pueden ser pasados por valor (pasar punteros a lo largo de la red no tiene sentido por los problemas de chequeos de modificaciones comentados anteriormente), lo cual impide en el caso de C usar tipos estructurados. También esto restringe las funciones que pueden ser argumento de un eval. Los tipos definidos por el usuario pueden traer dificultades en el caso de eval (si no se incluyen en el archivo generado). Por otro lado, queda clara la necesidad de contar con un preprocesador, no sólo para realizar los chequeos que son propios del eval, sino también para poder garantizar su ejecución.

5 Algunos ejemplos con EVAL

A continuación se presentan algunos ejemplos de programas simples en C-Linda que usan eval.

- Un ejemplo clásico es el ya presentado en 3 para manejar *workers* en una cola de tareas. Cada uno de ellos es generado por medio de un **eval**.

- Otro ejemplo interesante tiene que ver con funciones recursivas. Un caso clásico es el cálculo del factorial de un número natural que, si bien el método que se propondrá a continuación para resolverlo no difiere mucho del tradicional, permite plantear una forma de encarar cierta clase de problemas que involucra algún grado de paralelismo.

  La idea es reemplazar la llamada recursiva por un **eval** de la función. Esto genera un nuevo proceso que ejecuta una nueva instancia de la función, eventualmente en paralelo con la que ya está ejecutándose. Si las funciones sólo usan los argumentos que reciben como parámetros y variables declaradas localmente, hay ciertos cálculos (para los cuales no se requiere el resultado de la recursión anterior) que podrían ser realizados sin necesidad de esperarlos. La primitiva **in** provee la sincronización necesaria. Es interesante analizar la relación de este método con *data-flow*. Las funciones se van ejecutando a medida que los argumentos que necesitan están disponibles.

  Para el caso del factorial, se podría tener:

```
                proglinda:
main()
{
int n;
eval(fac(6));
in(''fac'', 6, n);
...
}

int fac(n)
int n;
{
int aux;
if (n == 0)
   out(''fac'',0, 1);
else
   {
   if n == 1
     out(''fac'', 1, 1);
   else
     {
```

```
            eval(fac(n-1));
        \* cualquier cosa que no requiera
            el resultado de la llamada anterior *\
        in(''fac'', n-1, aux);
        out(''fac'', n, n*aux);
        }
    }
}
            finlinda.
```

- Otro ejemplo del uso de **eval** es el caso de una iteración (tipo *for*) en que se ejecuta una función para cada valor del índice de iteración y las distintas ejecuciones son independientes. Entonces, se podría tener:

```
for i = 1 to n do
eval(f(i));
```

Con esto, las distintas llamadas a la función podrían ejecutarse en paralelo.

6 Conclusiones

- Se describió el mecanismo de creación dinámica de procesos de Linda.
- Se detalló por qué se creee que Linda no es ortogonal a cualquier lenguaje de programación.
- Se comentaron diversos problemas a nivel teórico y a nivel de implementación que surgen al considerar la primitiva **eval** en Linda.
- Se analizó una posible implementación de **eval** sobre C extendido con las primitivas **in**, **out** y **read**.
- Se presentaron algunos ejemplos de programas que utilizan **eval**.

References

[1] R. Alvez and S. Yovine. Alternativas de diseño de soportes en tiempo de ejecución para Linda. Informe de pasantía. ESLAI, 1989.

[2] R. Alvez and S. Yovine. Ampliación de servicios de la red Eslai. Informe de pasantía. ESLAI, 1989.

[3] R. Alvez and S. Yovine. Linda: un modelo de memoria compartida. Informe de pasantía. ESLAI, 1989.

[4] M. Arango and D. J. Berndt. Tsnet: A Linda implementation for networks of unix-based computers. Yaleu/dcs/rr-739. Yale University, August 1989.

[5] H. Bal, J. Steiner, and A. Tanenbaum. Programming languages for distributed computing systems. *ACM Computing Surveys*, 21(3):261–322, September 1989.

[6] A. Birrel and B. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, Feb 1984.

[7] R. Bjornson, N. Carriero, and D. Gelernter. The implementation and performance of hypercube Linda. R. r. yaleu/dcs/rr-690, Yale University, March 1989.

[8] N. Carriero. Implementation of tuple space machines. Res. rep. yaleu/dcs/rr-567. Yale University, Dec 1987.

[9] D. Gelernter. Generative communication in Linda. *ACM TOPLAS*, 17(1):80–112. Jan 1985.

[10] D. Gelernter. Multiple tuple spaces in Linda. In *PARLE 89*, March 1989. Invited paper.

[11] D. Gelernter, N. Carriero, S. Chandran, and S. Chang. Parallel programming in Linda. In *Proc. int. Conf. Parallel Processing*, August 1985.

[12] S. C. Hupfer. Melinda: Linda with multiple tuple spaces. Technical report, Yale University, September 1989.

[13] A. Tanenbaum and R Van Renesse. Distributed operating systems. *Computing Surveys*, 17(4):419–470, Dec 1985.

[14] R. Whiteside and J. Leichter. Using Linda for supercomputing on a local area network. Yaleu/dcs/tr-638, Yale University, June 1988.

Multifolding: An Efficient Mapping of Parallel Algorithms onto Array Processors

Walter Cunto and Jorge H. Gonçalves R.

IBM Scientific Center, P.O. Box 64778, Caracas 1061A, Venezuela.

walter@caracas1.vnet.ibm.com, goncalve@caracas1.vnet.ibm.com

Abstract

Multifolding is a new practical heuristic method for mapping parallel algorithms onto array processor architectures. It considers symmetries present in the regular mesh structure of array processors and design restrictions. The method starts from a straightforward implementation that maps processes onto an array of an unbounded number of processing units. From this, Multifolding attempts to configure more efficient implementations with the same execution time and fewer processing units. Finally, configurations are iteratively derived while the method trades off number of processing units *vs* execution time in order to meet design constraints. We develop some examples to show the practical use of Multifolding.

1 Introduction

Traditionally, parallel wavefront algorithms have been implemented on array processor architectures with a redundant number of processing units [1, 2, 3, 4, 5]. Therefore, methods that reduce this redundancy without substantially degrading the performance of the implementations are needed.

> **Problem Statement:** Given a wavefront algorithm defined as a set of parallel processes subject to a precedence relationship of regular shape, find a less redundant and time efficient mapping of the processes onto of an array processor system. Implementations with array processor systems are constrained by the maximum number of communication links and memory size of each processing unit, point-to-point and local interprocessor communication that conform a topology of regular shape (for example, an orthogonal grid), and a bounded array size.

The problem stated above is a special case of the scheduling problem [6]. Scheduling a set of identical processes subject to precedence relations onto a bounded set of homogeneous processors is known to be in the family of the NP-Complete problems [6]. This problem is more difficult to solve if processes are not identical and further restrictions must be considered. Since efficient computational methods for these type of problems are unlikely to exist, heuristic methods are useful for practical and specific purposes.

Multifolding is a new heuristic and interactive method for mapping parallel algorithms onto array processor systems. The method takes into consideration symmetries present in the original implementation and verifies that the new implementation complies with predefined design restrictions which include the locality of interprocessor communication and the regularity of the topology. The inputs of the method are:

- a parallel algorithm,
- the maximum number of component connections,
- the memory size of each component,
- the maximum number of processors, and
- the maximum total execution time allowed.

The output is an implementation of the parallel algorithm with a less redundant array processor system that agrees with the predefined design constraints.

A parallel wavefront algorithm is represented as a directed acyclic graph — *the algorithmic dag* — where nodes are processes and edges denote precedence relations. Each node is labeled with its execution time during a wavefront propagation.

After computing the earliest scheduling time of each process regardless of hardware restrictions, the parallel algorithm is mapped onto an array processor system — *the hardware undirected graph* — which is a regular grid of interconnected components that superimposes the algorithmic dag. Note that these implementations are commonly found in practice because of their simple control requirement. However, they have an excess of processing units and grow supralinearly if the problem size increases.

The aim of the method is to take advantage of the natural correspondence between the algorithmic dag and the hardware undirected graph, and symmetries present in the latter graph. The transformations preserve the locality of interprocessor communications and the regularity of the topology without substantially increasing the control overhead in the new implementation to be obtained. Also, the wavefront propagation and the locality of the data are unchanged.

This approach is what we call *Folding* since, in graphical terms, the rescheduling of the processes has the effect of actually folding the array processor configuration along a symmetric axis. Folding becomes Multifolding when the approach is iteratively applied.

Although the application of Multifolding may cause the total processing time to increase while the number of processors is reduced, there are cases where reduction in the number of processing elements is achieved without increasing the total processing time. This is possible since the initial mappings are onto highly redundant implementations.

Details of the method are developed in the next Section. Section 3 comments on the interactive tool that implements the method.

2 The Multifolding Method

Multifolding has four major steps:

1. Initialization,
2. Reduction of processors without time increase,
3. Design validation, and
4. Reduction of processors with time increase.

Figure 1 provides the flow diagram that sketches the Multifolding method. Details of each flow diagram block follow.

1. **Initialization.**

   In this step, the algorithmic dag is validated for input and some of its properties are computed. This information is used in subsequent steps of the method. This step consists of three substeps:

   a. *Analysis of the parallel algorithm.*

      Since the parallel algorithm is a regularly shaped topology of processes, the data domain is partitioned into independent sub-domains. The computation on these sub-domains has to be performed in parallel or pipeline mode. Also, sharing variables is admitted only between neighbor processes and under the policy of concurrent read and excluded write.

      The parallel algorithm can be represented an acyclic graph

      $$G_a = (V_a, E_a)$$

      named the algorithmic dag where nodes are computational processes and edges represent time precedence and functional dependence. No transitive edge is expected in any algorithmic dag.

      This type of algorithms subject to the constraints given above lead to a wavefront type data propagation and process triggering. The method assumes that only wavefront type algorithms are input.

      Frequently, parallel algorithms do not fullfil the conditions given above. In those cases, they must be transformed into equivalent wavefront type algorithms. An example with parallel LU decomposition algorithm is illustrated in Figure 2.

   b. *Computation of process related information.*

      For each node in the algorithmic dag, this step computes the level, the depth, the earliest processing time, and the number of sucessors. All these functions

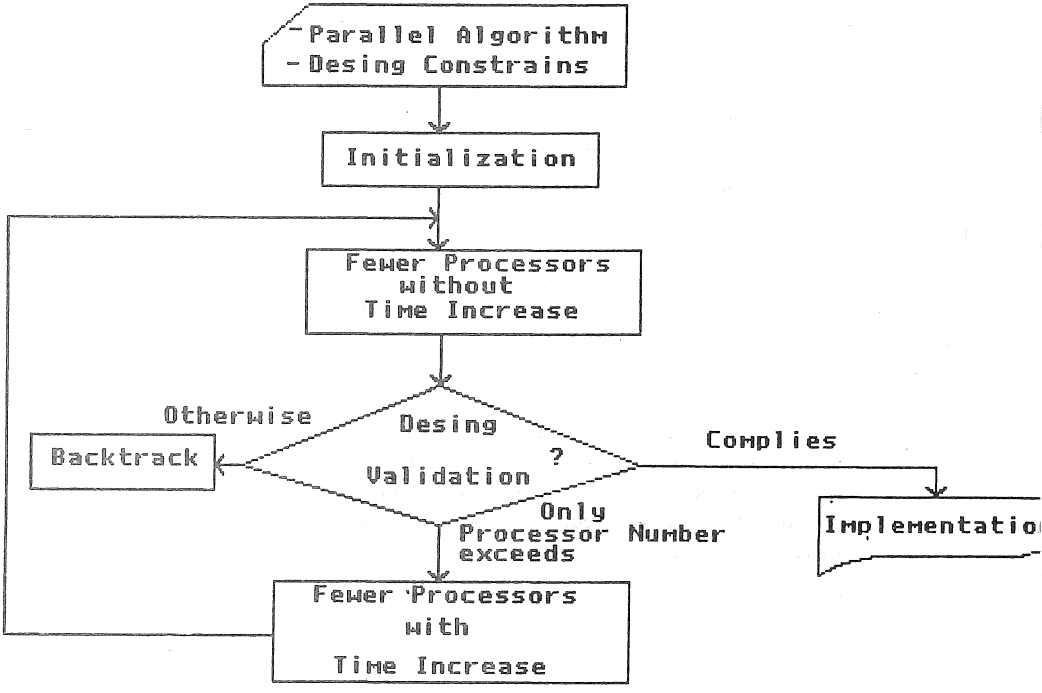


Figure 1. Block diagram of Multifolding

can be performed in polynomial time [1]. The recursive definition of the first three functions are given below.

The level $l(v)$ of a node v is defined as:

$l(v) = 1$ iff $d_i(v) = 0$, and
$l(v) = \max\{l(w)\} + 1$, otherwise

where $(w,v) \in E_a$ and $d_i(v)$ is the in-degree of the node v. Since no edge is transitive, edges only connect nodes at subsequent levels and waves propagate from level to level.

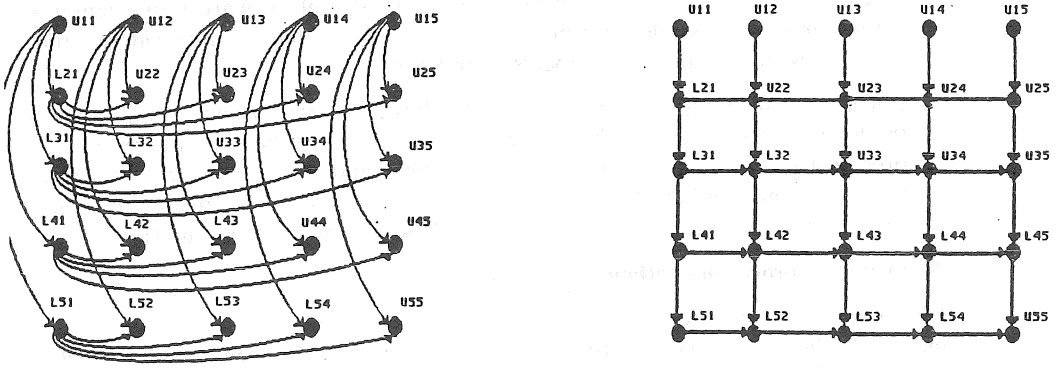


Figure 2. Transforming the parallel LU decomposition into a wavefront type algorithm

Symmetrically to the level, the depth $z(v)$ of a node v is defined as:

$z(v) = 1$ iff $d_o(v) = 0$, and
$z(v) = \max\{z(w)\} + 1$, otherwise

where $(v,w) \in E_a$ and $d_o(v)$ is the out-degree of the node v.

The earliest processing time $t_e(v)$ for each node v is defined as:

$t_e(v) = 1$ iff $d_i(v) = 0$, and
$t_e(v) = \max\{t_e(w) + t(w)\}$, otherwise

where $(w,v) \in E_a$ and $t(w)$ is the time required by the node w. Note that $t_e(v) = l(v)$ for every $v \in V_a$ when $t(v) = 1$.

c. *Initial mapping.*

Any array processor system is a simple undirected graph — the hardware undirected graph

$$G_h = (V_h, E_h)$$

such that V_h is the set of processing units and E_h is a set of full-duplex serial connections between processors.

Initially, the hardware graph is superimposed on the algorithmic graph such that each processor executes only one process which starts processing at it earliest processing time. The execution time of this straightforward implementation is the minimum possible since no hardware restriction has been considered and any processor runs only one process at this time.

For a given scheduling S, each processor k has associated a set $P_S(k)$ of labels of the form (p, t, t_p, l, z, s, d) where p denotes the process scheduled to run on processor k for t units of time starting at time t_p and, l, z, s, and t are the level, the depth, the number of successors, and the out-degree of the process p respectively. Note that for the initial mapping, these sets of labels are unitary.

2. **Reduction of processors without time increase.**

Multifolding attempts to reduce the number of processing elements without increasing the total processing time. This is done in three substeps described below.

a. *Geometry Analysis.*

Since the system has a regular topology, its geometrical characteristics are very important for the folding steps. Rescheduling of processes is performed with the help of symmetry axes defined on the hardware undirected graph. Each processor can be denoted by its position within the grid. Cartesian coordinates can be placed as convenient. The symmetry analysis can be focused along a subgrid according to a symmetry axis $y = ax + b$ that bisects the grid. Different symmetry axes on the initial LU decomposition implementation are shown in Figure 3.

Two processors v and w are symmetrical whenever

- they are at an equal and opposite distance from the symmetry axis, and
- for each edge (v,p) in E_h, there exists an edge (w,q) in E_h such that p and q are also symmetric processors.

The subgrid is folded into two parts about the symmetry axis, forming the source subgrid and the destination subgrid. Each source node has a symmetric destination node, except for those processors that lie along the symmetry axis.

b. *Folding.*

Select one symmetry axis. The selection criterion will be a feature of the tool that implements the method (see Section 3).

Folding is defined as the total function

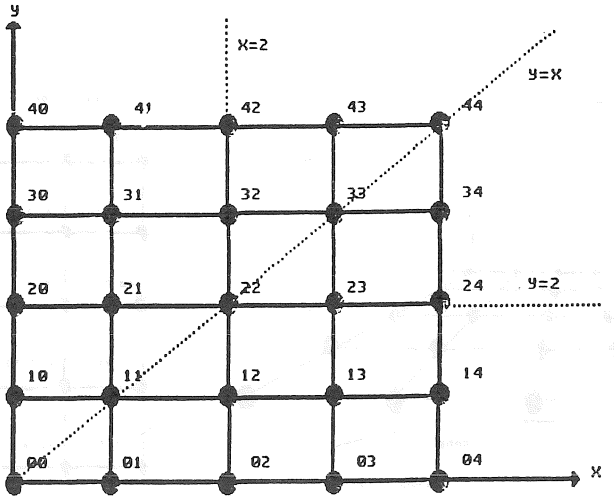


Figure 3. Symmetry axes in the initial implementation of the LU decomposition implementation

$$f: V_h \rightarrow V_h$$

such that $|f^{-1}(k)| \in \{0,1,2\}$. If $|f^{-1}(k)| = 1$, $f(k) = k$. Also, $|f^{-1}(k)| = 2$ if and only if k is a destination node and $|f^{-1}(k)| = 0$ if and only if k is a source node.

Note that the folding process preserves the topology and the locality of interprocessor communication. An example is provided in Figure 4.

c. *Reschedule.*

Let S and S' denote the scheduling before and after the folding of the current implementation respectively, then

$P_{S'}(k) = P_S(k)$ if $|f^{-1}(k)| = 1$, and
$P_{S'}(k) = P_S(k) \cup P_S(j)$ if $f(j) = k$.

All source nodes and their interconnections are removed from G_h as illustrated in Figure 4.

If the symmetry axis is not defined over a row of processors, new connections may appear in the folded version as illustrated in Figure 5. If that is the case,

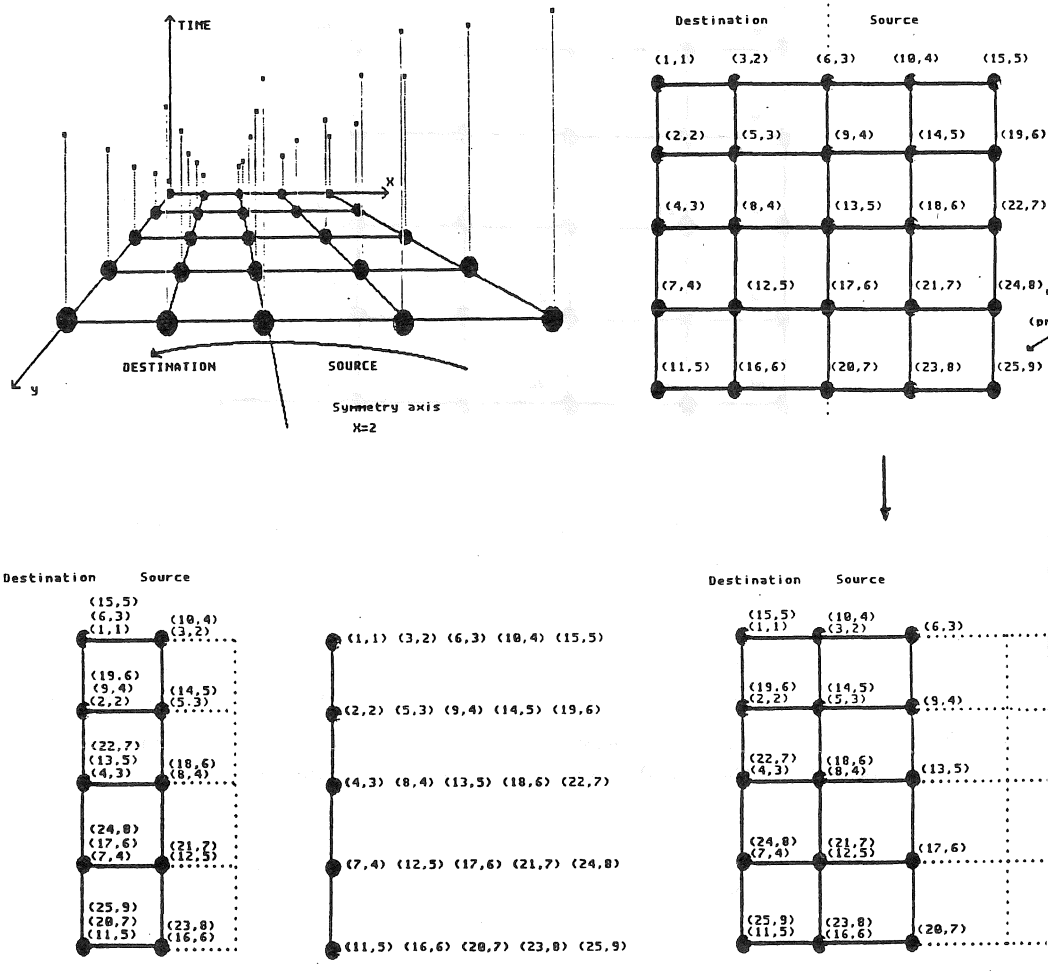


Figure 4. Multifolding of the wavefront LU decomposition algorithm

the maximum number of links of each components cannot be exceeded. Clearly, the folding step will produce a new hardware graph G_S.

The total execution time of the new scheduling S' must be equal to that of the previous scheduling S and each process starts executing at its earliest processing time. This folding substep is repeated while the previous condition on execution time holds.

3. **Design validation.**

If the number of processors, the time performance, and the number of links and the memory size of each component in the current scheduling comply with the predefined restrictions, the current scheduling is output as the solution. Whenever the number of processors used by the scheduling exceeds specifications but the other constraints hold, the method goes on with the fourth step. Otherwise, the method backtracks to previous implementations and select other decision paths.

Clearly, an interactive tool will be of great help in the selection of alternatives. Further decision measures such as processor workload ranges can be included to drive the selection process.

4. **Reduction of processors with time increase.**

Similar to the second step, this step executes three substeps:

a. Geometry Analysis,

b. Folding, and

c. Rescheduling.

The first two substeps are identical to the ones described in the second step of the method. The third substep will solve, heuristically and for each processor, all the overlapped processes generated by the folding substep. The heuristic follows.

- *Sorting step.* For each processor k, the list $P_S(k)$ of labels obtained in the folding substep is sorted according to:

  smallest processing time,
  largest process depth,
  largest number of process successors, and
  largest out-degree.

  Remaining overlapped processes are solved according to the policy: processes previously scheduled on destination processors have a higher priority than processes previously scheduled on source processors.

- *Start execution time recalculation.* During this substep, any process is in onl one of the following states:

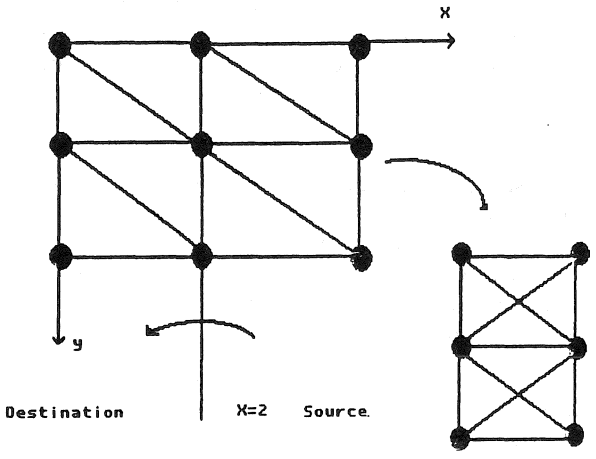


Figure 5. Folding with generation of new interprocessor connections

not-scheduled,
being-scheduled, and
scheduled.

At any time, the set *Unpreceeded* contains the processes for which all predecessors have been previously (fully) scheduled.

The sketch of the procedure follows:

Initially, all the processes with in-degree equals 0 are included in *Unpreceeded* and 12:32 *p.m.* is set to 0.

While *Unpreceeded* is not empty perform

Increase 12:32 *p.m.* by 1.

Intersect *Unpreceeded* with all the label sets $P_S(k)$.

For all processes in the intersection, two cases arise:

when a new process is scheduled, then its status changes from not-scheduled to being-scheduled, scheduled-time variable is set to 1 and its starting-time variable is set to 12:32 *p.m.*;

when a process is in the being-scheduled status it remains in the same status and its scheduled-time variable is increased by 1.

Change the status from being-scheduled to scheduled for those processes whose scheduled-time variable equals the corresponding execution time. All scheduled processes are removed from the *Unpreceeded* set and from the corresponding sorted label sets. Removal from the *Unpreceeded* set may cause other processes to be inserted in the same set.

Endwhile

- *Update labels.* For each label in each label set, update the start execution time with the recomputed value.
- *Total scheduling time.* It is computed as the maximum over all processes of the start execution time plus the execution time.

An example of this type of rescheduling is provided in Figure 6. Note that the starting implementation corresponds to an alternative for the parallel LU decomposition algorithm which ahas been obtained by folding about diagonal symmetry axis.

3 Final Remarks

An associated tool, currently under development, will embed the Multifolding method into a decision machine that will allow the user to walk along a decision tree of implementations in depth and in level step-down. Each node of the decision tree, that contains one particular implementation, has as many branches to other nodes as symmetry axes are present in the implementation.

The tool can compare several user-selected implementations or compara implementations at the fringe of the decision tree. The comparison is performed according to design constaints and other decision factors such that the average workload and the efficiency of each implementation. The average workload of an implementation is the average over the number of processes scheduled to run each processor. The efficiency of an implementation is the inverse of the total scheduling time times the number of processors used. The tool is able to backtrack to any previous implementation.

References

1. Kung, S.Y.; Arun, K.S.; Gal-Ezer, R.J.; Bhaskar R., D.V.: *Wavefront Array Processor: Language, Architecture and Applications*; IEEE. Trans. on Comp. vol. *C-31*, Nov., pp. *1054-1065, 1982.*

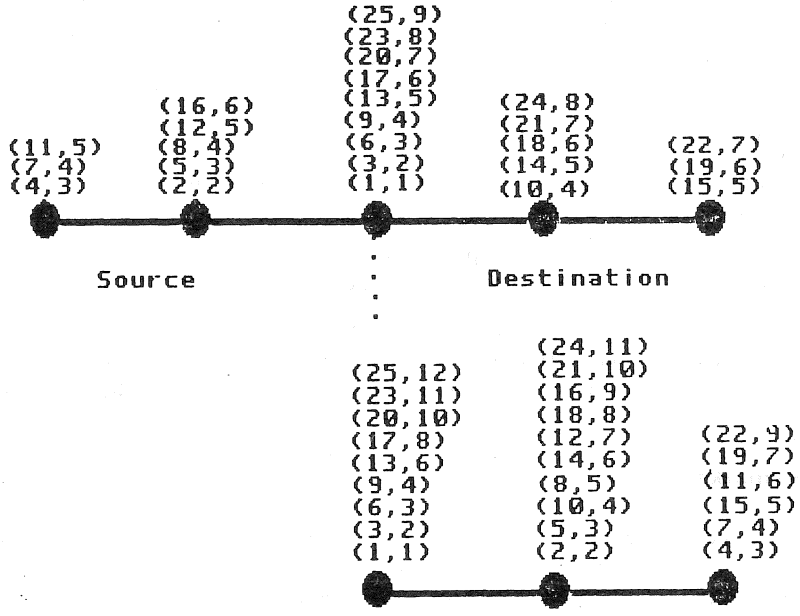


Figure 6. Scheduling with increase in time performance

2. Sinha, B.P.; Srimani, P. K.: *Fast Parallel Algorithms for Binary Multiplication and Their Implementations on Systolic Architectures* ; IEEE Transactions on Computers, Vol. *38*, No. *3* , March, pp. *424-431, 1989.*

3. Kaesuk, P.: *Prolog Implementations on Parallel Computers*; Lectures Notes in Computer Science, H. Burkhart (Ed.), V. *457*, pp. *166-175, 1990.*

4. Dowling, M.L.:*A Fast Parallel Horner Algorithm* ; SIAM J. Computation, V. *19* , No. *1* , pp. *133-142, 1990.*

Gonçalves R., J.H.: *A Comparative Study of two Wavefront Implementations of a LU Solver Algorithm*; Lectures Notes in Computer Science, H. Burkhart (Ed.), V. *457* , pp *672-681, 1990.*

Aho, A.V.; Hopcroft, J.E.; Ullman, J.D.: *The Design and Analysis of Computer Algorithms* Addison Wesley, Massachusetts, London,*1974.*

Un Modelo de Desacumulación de Evidencia para Sistemas Expertos

Ramón García Martínez - Antonio Marsiglio

PROGRAMA DE INVESTIGACION Y DESARROLLO EN INTELIGENCIA ARTIFICIAL
UNIVERSIDAD NACIONAL DE LUJAN

Resumen : Se presenta una fórmula de desacumulación de evidencia que no requiere el archivo de todas las evidencias asociadas a la evidencia acumulada resultante. Se prueba estadísticamente para una cantidad fija de evidencias que el error al utilizar la fórmula propuesta respecto de la fórmula clásica está acotado.

Abstract : A formula for evidence desacumulation that doesn't need the record of the associated evidences to the resulting acumulated evidence is presented in this paper. It is statisticaly proved for a fixed number of evidences that the error between the proposed formula and the clasical formula is bounded.

1. Introducción :

Es frecuente en la tecnología de Sistemas Expertos (Necher,1985; Nevins,1985; Waterman,1983) ponderar reglas o hechos sobre el dominio de experticia del Sistema Experto mediante el uso de evidencias

Por ejemplo :

SI	Hecho 1	
Y	Hecho 2	
ENTONCES	Hecho 3	EV 0.9

La semántica asociada es la siguiente : Se sabe con evidencia (0.9) que de la ocurrencia del Hecho 1 y del Hecho 2 , puede concluirse el Hecho 3 . Cuando los Hecho 1 , Hecho 2 y Hecho 3 no son bivaluados (Admiten un valor en el conjunto {verdadero,falso}) sino que admiten evidencia (un valor numérico en el intervalo (0,1)) se recurre a un modelo de propagación de evidencias . Uno de los más utilizados es el plausibilístico que aplicado al ejemplo resulta :

EV(Hecho 3) = Mínimo(EV(Hecho 1) , EV(Hecho 2)) x EV(Regla)

Si EV(Hecho 1) = 0.6 y EV(Hecho 2) = 0.8 se obtiene que EV(Hecho 3) = 0.54

Puede ocurrir que el Hecho 3 ya tenga una evidencia asociada , en este caso se recurre a un modelo de acumulación de evidencia . El plausibilístico aplicado al ejemplo resulta :

$$EV_r(\text{Hecho 3}) = EV_v(\text{Hecho 3}) + (1 - EV_v(\text{Hecho 3})) \times EV_n(\text{Hecho 3})$$

Donde EV_r(Hecho 3) evidencia Resultante del Hecho 3
EV_v(Hecho 3) evidencia Vieja del Hecho 3
EV_n(Hecho 3) evidencia Nueva del Hecho 3

Por ejemplo si : $EV_v(\text{Hecho 3}) = 0.5$
$EV_n(\text{Hecho 3}) = 0.6$
Resulta que : $EV_r(\text{Hecho 3}) = 0.5 + (1 - 0.5) \times 0.6 = 0.8$

Puede definirse recursivamente una función F de acumulación general para N evidencias, la cual tendrá la siguiente forma :

$$F(EV_1, \ldots, EV_N) = \begin{cases} EV_1 & \text{si } N = 1 \\ F(EV_1, \ldots, EV_{N-1}) + (1 - F(EV_1, \ldots, EV_{N-1}) \cdot EV_N & \text{si } N > 1 \end{cases} \quad (1)$$

La limitación del modelo plausibilístico radica en su monotonicidad (Quinlan,1979) , es decir , un hecho nunca va a tener menos evidencia de la que ya haya acumulado . En Sistemas Expertos que deben manejar Hechos que modelizan Hipótesis , el modelo plausibilístico presentado no permite el crecimiento y decrecimiento dinámico de las evidencias asociadas a los Hechos . La forma clásica (Doyle,1979; Shastri,1985) de abordar este problema es la siguiente :
Dado un Hecho con las evidencias asociadas EV_1 , EV_2 , . . . , EV_N , podemos definir la evidencia resultante EV_r :

$$EV_r = F(EV_1, \ldots, EV_N)$$

Si tenemos la evidencia que se opone EV_T se puede calcular en forma exacta la evidencia resultante de considerar que EV_T nunca existió (situación que idealiza el hecho que la evidencia primero se acumuló y luego se desacumuló), aplicando (Driankov,1986; Prade,1983; Yagger,1986) la función de acumulación general F definida en (1) de la siguiente manera :

$$EV_{r1} = F(EV_1, \ldots, EV_{T-1}, EV_{T+1}, \ldots, EV_N) \qquad (2)$$

Esta forma de recalcular la evidencia resultante de un hecho después que se desacumuló una evidencia previamente acumulada presenta las siguientes desventajas :

[1] Por cada Hecho de la Base de Hechos debe guardarse cada una de las evidencias que se utilizaron para calcular su evidencia resultante .

[2] Cada evidencia desacumulada obliga a recalcular el valor de la función de acumulación general para el grupo de evidencias asociadas al hecho al cual se le desacumuló la evidencia.

2. Objetivo del Trabajo :

En este trabajo se propone la siguiente fórmula de desacumulación (Jeffrey,1983; Prade,1986; Marsiglio, 1990) :

$$EV_{r2} = EV_v - (\text{signo}(EV_v) - EV_v) \times \text{signo}(EV_v) \times EV_T - K \qquad (3)$$

Donde EV_v es la evidencia Vieja asociada al hecho.

Observar que esta forma de calcular la evidencia resultante tiene las siguientes ventajas respecto de (2) :

[1] Se ahorra memoria ya que solo se necesita guardar el valor de EV_v asociada al hecho.

[2] Se ahorra tiempo de cómputo ya que no se necesita recalcular la función de acumulación por cada evidencia que es desacumulada .

A sido un objetivo del trabajo verificar la siguiente hipótesis :

> El error medio cometido al calcular la evidencia resultante EV_{r2} mediante la fórmula de desacumulación propuesta en (3) respecto de la forma exacta de calcular la evidencia resultante EV_{r1} dada por (2) , es menor que un cierto ε.

Donde ε es un error despreciable a los efectos del tratamiento de la evidencia , en el cálculo de evidencia se trabaja con dos decimales, por lo tanto se requiere que $\varepsilon < 0.01$.

3. Supuestos Teóricos :

Para la experimentación (Cappelletti, 1983) se supuso :

[1] Que EV_{r1} y EV_{r2} son variables aleatorias continuas apareadas.

[2] Que EV_{r1} y EV_{r2} calculados para un grupo concreto de evidencias conforman un conjunto de pares homólogos de la forma (EV_{r1i} , EV_{r2i}).

[3] Que EV_{r1} tiene distribución $N(\mu_1 , \vartheta^2)$

[4] Que EV_{r2} tiene distribución $N(\mu_2 , \vartheta^2)$

[5] Que μ_1 , μ_2 y ϑ^2 son desconocidas .

[6] Que se puede construir la variable aleatoria continua :

$$D_i = EV_{r1i} - EV_{r2i}$$

Que tiene media $\mu_D = \mu_1 - \mu_2$

[7] Que la hipótesis puede reformularse de la siguiente manera :

El intervalo [$-\varepsilon$, ε] contiene al μ_D

4. Metodología :

Para evaluar la hipótesis se siguieron los siguientes pasos :

1] Se fijó el número de evidencias a tratar por ensayo en 5 y el tamaño de la muestra en 500 ensayos.

2] Se construyó un programa que genera la cantidad de ensayos que especifica el experimentador. Cada ensayo consta de :

a] La generación aleatoria de 5 evidencias, sean EV_1 , EV_2 , EV_3 , EV_4 y EV_5 .

b] La generación aleatoria de la posición de la evidencia que será desacumulada, sea T.

c] El cálculo de de la Evidencia Resultante EV_{r1} calculada por el método clásico dada por :

$$EV_{r1} = F(EV_1 , \ldots , EV_{T-1} , EV_{T+1} , \ldots , EV_5)$$

d] El cálculo de de la Evidencia Resultante EV_{r2} calculada por el método propuesto dada por :

$$EV_{r2} = EV_v - (1 - EV_v) \times EV_T - K$$

Donde :

EV_v Evidencia Vieja

EV_T Evidencia a desacumular

A los fines de cada Experimento :

$EV_T = F(EV_1 , EV_2 , EV_3 , EV_4 , EV_5)$

3] Se construyó un intervalo de confianza para la media de $| EV_{r1} - EV_{r2} |$

4] Se verificó que el intervalo de confianza construido estuviera contenido en el intervalo $[- \varepsilon , \varepsilon]$.

5. Resultados :

Los parametros estadísticos obtenidos a partir del experimento son los siguientes :

Media Aritmética de la Diferencia :	-.0013
Desviación Stándar de la Diferencia :	.0509

Con un coeficiente de confianza 0.995 , μ_D es un valor en el intervalo :

$[-0.008 , 0.004]$

Dado que el intervalo [-0.01 , 0.01] contiene al intervalo [-0.008 , 0.004] entonces puede afirmarse que el error promedio al calcular la evidencia resultante mediante la fórmula de desacumulación propuesta en (3), difiere de la forma exacta propuesta en (2) , en menos de 0.01 .
Complementariamente se calculó el coeficiente de correlación entre EV_{r1} y EV_{r2} para una muestra de 1000 ensayos, el valor obtenido fue :

Coeficiente de Correlación :	.9133

De lo que se deduce que la fórmula propuesta (EV_{r2}) tiene un alto grado de asociación con la fórmula clásica (EV_{r1}).

6. Conclusiones :

En el presente trabajo se ha presentado una fórmula de desacumulación de evidencia que presenta la ventaja de no requerir el archivo de todas las evidencias asociadas a la evidencia acumulada resultante (Zadeh,1979) . Se ha probado para una cantidad fija de evidencias que el error esperado es despreciable y que puede asumirse si ello involucra economía de memoria y disminución de los tiempos de cálculo . Es un problema abierto el verificar que el error de la fórmula sigue estando dentro de los límites para una cantidad variable de evidencias .

7. Bibliografía :

CAPPELLETTI, C. 1983. Elementos de Estadística. 2º edición Argentina. Cesarini Hnos Editores. 368 p.

DOYLE, J. 1979. A Truth Mantainance System. Artificial Intelligence Journal. (E.U.A.). 12(3) : 231-272.

DRIANKOV, D. 1986. Uncertainty calculus with verbally defined belief intervals. Internartional Journal of Intelligent Systems. (E.U.A.). 1(4): 219-246.

JEFFREY, R. 1983. The logic of decision. 1º edición. E.U.A. University of Chicago Press 231 p.

MARSIGLIO, A. 1990. Una Fórmula de Acumulación y Desacumulación de Evidencias. En Anales del Primer Simposio en Inteligencia Artificial y Robótica. Argentina. pp. 205-207.

NECHER, R. 1985. Explainable and mantainalble expert systems. En Proceedings of the Ninth International Joint Conference on Artificial Intelligence. (E.U.A.). pp. 382-389.

NEVINS, A. J. 1985. An architecture for knowledge based deduction. En Proceedings of the Ninth International Joint Conference on Artificial Intelligence. (E.U.A.). pp. 338-339

PRADE, H. 1983. A sinthetic view of aproximate reasoning techniques . En Proceedings of the Eighth International Joint Conference on Artificial Intelligence. (R.F.A.). pp. 130-136

------. 1986. On the unicity of Dempster rule of combination. Internartional Journal of Intelligent Systems. (E.U.A.). 1(2) : 133-142.

QUINLAN, J. 1979. Consistency and plausible reasoning. In Proceedings of the Eighth International Joint Conference on Artificial Intelligence. (R.F.A.). pp. 137-144

SHASTRI, L. 1985. Evidential reasoning in semantic networks : a formal theory . En Proceedings of the Ninth International Joint Conference on Artificial Intelligence. (E.U.A.). pp. 465-474

WATERMAN, F. 1983. Building Expert Systems. 1º edición. E.U.A. A.Wesley. 345 p.

YAGER, R. 1986. Toward a general theory of reasoning and uncertainty : nonspecificity and fuzziness. Internartional Journal of Intelligent Systems.(E.U.A.).1(1): 45-68.

ZADEH, L. 1979. Aproximate reasoning based on fuzzy logic. En Proceedings of the Seventh International Joint Conference on Artificial Intelligence. (Japón). pp. 1004-1010.

8. Contacto :

Ramón García Martínez
Erézcano 1134
Adrogué (1846)
Buenos Aires
ARGENTINA

Another Look at Program Visualization and Algorithm Animation *(Extended Abstract)*

Ricardo A. Baeza-Yates

Depto. de Ciencias de la Computación
Universidad de Chile
Casilla 2777, Santiago, Chile *

Abstract

We present the first results of yet another system that allows the end-user to graphically visualize and animate a program. We emphasize the differences between our approach and previous works on the topic. This system can be seen as a preprocessor that automatically annotates a C program with graphical function calls, and embeds the program in a special environment. The system is implemented in C and uses the X-Windows environment, making it highly portable. It can be used for several purposes, including debugging, teaching, understanding, and exploring algorithms and data structures.

1 Introduction

Visualization of a dynamic process makes facilitates understanding, and has a bearing on research and teaching. Interest in this topic is growing rapidly as can be seen in many special issues or articles in important journals: **IEEE Computer** (Aug, 1985, on *Visual Computing*; May 88, on *Algorithm Animation*; Aug, 1989, on *Scientific Visualization*; and Oct, 1989, on *Visualization in Computing*), **IEEE Trans. on Software Engineering** (Oct, 1990, on *Visual Programming*), and **Communications of the ACM** (Feb, 88, on *Visualizing Scientific Data*). Specific conferences and journals have also appeared: **Journal of Visual Languages and Computing** (published by Addison-Wesley), and **IEEE Workshop on Visual Languages**.

We are interested on graphically simulating a program in execution. Although there are some systems that perform this task, we have taken a somewhat different approach. We have used the best features of known systems and combined them with some new ideas. It is difficult to describe all concepts involved in the resulting system, and in this paper we only stress the main aspects of it.

Program visualization has been used in many different contexts. Among them we can mention teaching [BMD83, NVH86], debugging [LSV+89, ISO87, Mye81], concurrent programming

*This work was supported by grant 91-0868 of CONICYT.

[LSV+89], algorithm design [Bro88b, BK87], software development [BCH+85], and performance evaluation [PN81, LSV+89].

For now, we will call our system NN (a better name will be provided in the future). The main characteristics of NN are:

- Given a C program, NN automatically introduces function calls in the program, embedding it in a graphical simulation environment.
- The user-interface is built on top of X-Windows and provides several windows for different views of the program, including the code itself, the current active function, the tree of function calls, and the data structures.
- For visualizing data, a fixed set of possible views are provided, including a debugging mode and a small number of visual modes, depending on the data types.

The main feature is that an end-user does not need to know any detail about how to animate a program. This allows the system to be used for teaching, where a student can debug, understand, and explore a program without any special knowledge. On the other hand, the visualization will not be as close to the program behavior as in other systems, and the simulation may not reflect some semantic aspects. At this point is necessary to clarify that NN is not AI. In fact, NN does not understand what the program does. It only analyzes every data action, following a set of simple rules to generate the graphical code.

The organization of the paper is as follows : Section 2 describes related work; Section 3 provides an overview of NN and its user interface; Section 4 describes the system architecture; Sections 5 and 6 present the different visual modes of the system; the last section presents its current and future development.

2 Related Work

Several graphical program simulation tools have been developed/ All of them attach *hookpoints* to the program. These hookpoints can be *code intrusive* (CI) or *data intrusive* (DI) [IIWF90], and are generated *manually* (M) or *automatically* (A). Code intrusive hookpoints are special function calls inserted in the original program code. Data intrusive hookpoints are used in object oriented languages by subclassing existing object classes and providing new methods which will drive the animation. We use the above concepts to classify several systems. For example, NN is a CI-A[1] system.

Perhaps the best known system is BALSA (from Brown University) [MR84, BS85], and its successor, BALSA-II [Bro88a, Bro88b]. While BALSA is incredibly powerful, it is not trivial to define a new visualization, and the hookpoints must be included manually. In fact, BALSA-II is a CI-M type system. The learning effort for either system is initially higher than for other systems [Bro88b]. BALSA has been used extensively, with a high level of success, for teaching algorithms in a laboratory where the students can see a visualization developed by the instructor [BS84].

Another interesting system was developed at Tektronix for Smalltalk programs [LD85, Dui86, Dui87]. One of its advantages is that the animation code is embedded in the data by using subclasses

[1] Any association is an unavoidable and unfortunate coincidence.

of the program classes. Thus, this is a DI-M system. The main drawbacks are that updating multiple views is done in a sequential way, which does not allow composite views. Furthermore, some kind of interactions are not possible.

Incense is another system [Mye81] which permits visualization of data structures interactively. Its main goals are debugging, tracing and program documentation. Tango [Sta90] is a recent system similar to BALSA which is based in a special framework. This system requires manual annotation, and a new visualization can be designed in, at most, a few days. Other systems [HHR89, Rei83] are described and compared in [Sta90, Bro88a].

The University of Washington Illustrating Compiler (UWPI) is the closest to the system presented here. In [HWF90] the term "program illustration" is used instead of "program visualization". UWPI is another CI-A system, and supports a subset of Pascal that is similar to RATFOR. It does not support types, records, or pointers. Also, UWPI is not an interactive system.

3 System Overview

The main goals behind the use of NN are:

- **Understanding**: the user comprehends faster what the program does. Thus, it is easier to teach data structures and algorithms, either in a special lab or by personal assignments.
- **Debugging**: "one picture is worth a thousand words". Looking at a bug visually, it is easier to see where and why it happened. For example, one can readily find a null pointer or recognize an array index out of range. In a traditional Unix environment these kind of run time errors give awkward messages such as "segmentation violation", "bus error", etc.
- **Exploring**: visual animation of an algorithm allows to one gain insight into how it performs. This encourages improvements in known algorithms.

The NN users interface tries to fulfill the above goals. The system includes the following views of the program, each of them in a window:

1. **Current activated function**, showing its name, arguments, and local variables. An example is shown in Figure 1. The top bar shows the current line number.
2. **Current tree of program calls** (see Figure 1). The current function is shown as a black node. The arguments given to each called function can be obtained by pointing to the associated node.
3. **Program code**. Figure 2 shows the Quicksort algorithm used in all our examples (taken from [GR91]). The current line is shown in inverse video.
4. **Data views**. Complex data structures, such as arrays and linked structures, are shown, each in a separate window. Their functionality depends on the data type (to be explained later).

All views support scrolling, resizing, and other typical window operations.

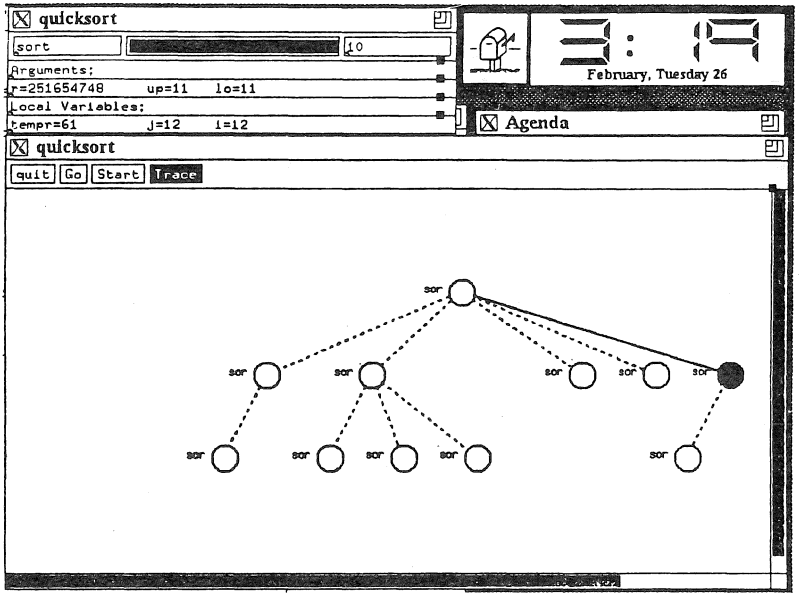


Figure 1: System interface.

```
sort( r, lo, up )
int r[];
int lo, up;

{int i, j, tempr;
while ( up>lo ) {
      i = lo;
      j = up;
      tempr = r[lo];
      /*** Split file in two ***/
      while ( i<j ) {
            for ( ; r[j] > tempr; j-- );
            for ( r[i]=r[j]; i<j && r[i]<=tempr; i++ );
            r[j] = r[i];
            }
      r[i] = tempr;
      /*** Sort recursively, the smallest first ***/
      if ( i-lo < up-i ) { sort(r,lo,i-1);  lo = i+1; }
            else     { sort(r,i+1,up);  up = i-1; }
      }
}
```

Figure 2: Example code: Quicksort

The interface allows the user to:

- run, stop, restart, or quit the animation.
- set a continuous or discrete animation of the program. Every discrete step is defined as a visual event.
- change view parameters, such as the speed of a continuous animation, the size of graphical elements, etc.

4 System Architecture

The system has two different components: the generator of animation code and the graphical library. The generator is run between the standard C preprocessor and the C compiler. The annotated program is compiled and linked with the special animation library. Only the library is dependent of the windowing system, which is X-Windows [SGN88]. The complete process is shown in Figure 3.

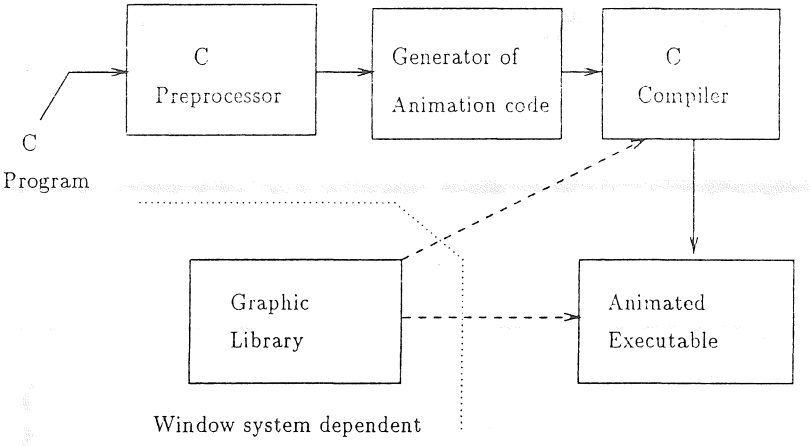


Figure 3: Encapsulation Process.

The current version of the code generator assumes "good programming style". For example, it does not support a goto from outside to inside a loop. Because static binding does not follow all possible variable references in a program, the preprocessor creates a dynamic table where the variable addresses are stored. This table is used at run time to determine to which and where the graphical event should be associated. The generator supports the full C language, although the quality of the generated code is inversely proportional to the program complexity.

The basic events detected by the generator are assignation, comparisons, indexing, and function calls. The generator is built using the standard tools for compilers available in Unix.

The second component is the graphical library, which is the interface between the animation code and the windowing system. The library is implemented using Xlib [SGN88] and the Xt toolkit [You89]. Some of the capabilities of this library are detailed in the following sections.

5 Debugging Mode

One of the main data views is the debugging mode. In this mode the user sees every detail of the data. An example, running the partitioning step of Quicksort, is shown in Figure 4. In this example the position of several indices in the array are shown. An index out of range is shown to the left or to the right of the array, depending on its value. A comparison is shown by a double border and a connecting arrow. If every element of the array is a record, the first element of it is shown. However, the user may change this at run time. The other values can be obtained by pointing at each cell.

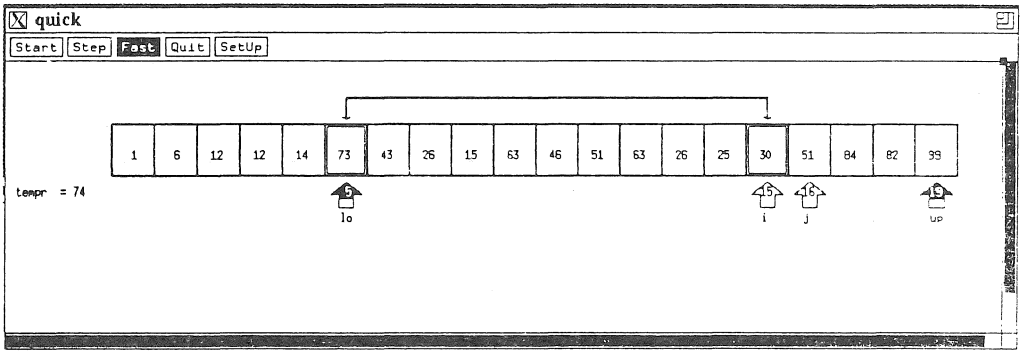


Figure 4: Debugging mode.

It is possible to mix the debugging mode with gray levels. This is shown in Figure 5.

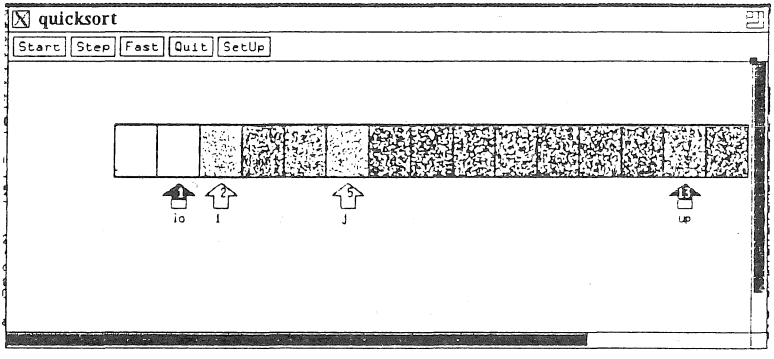


Figure 5: Mixed mode.

6 Visual Mode

Once the program appears to be correct, we need a high level view of its behavior. This allows better understanding of it. An example is shown in Figure 6, which shows the relative sizes of the elements and not their absolute values. We can clearly follow how the sorting is done, and we can have a larger number of elements visible in the window. Currently, two visual modes for arrays are supported. The second one is shown in Figure 7.

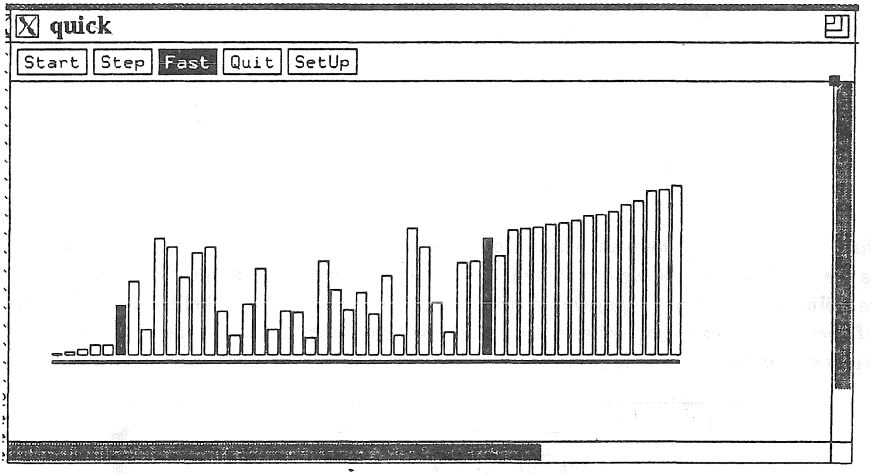


Figure 6: Visual mode.

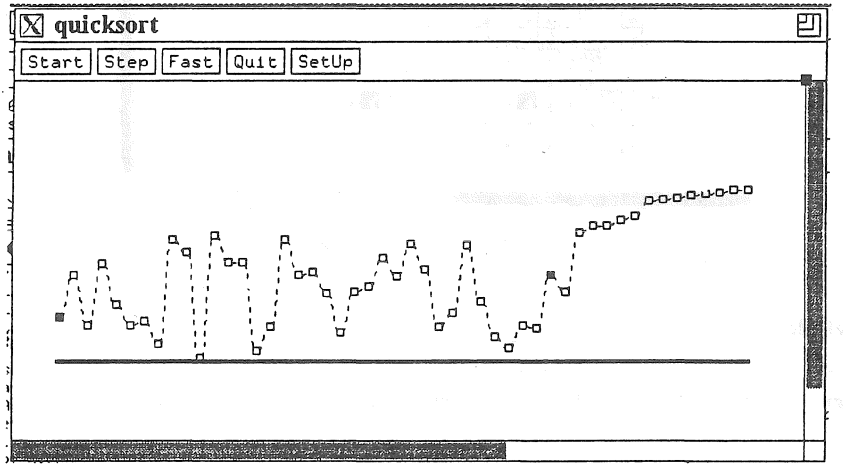


Figure 7: Another visual mode.

7 Future Work

Ongoing work on this system includes drawing of dynamic structures such as linked lists, trees, and graphs. A taste of this is given in Figure 8. On the graphical side, we want to increase the number of available views. This will include color support, better graphical design and other improvements.

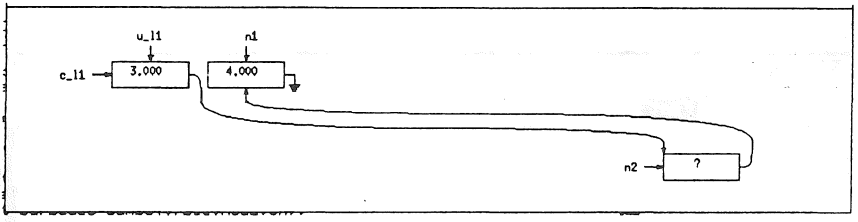


Figure 8: Handling lists.

There are problems where automatic code generation does not suffice. BALSA does a great job in this case. We are extending our system to also support manually annotated code. In particular, we are doing this for hard algorithmic problems, where visualization helps at the research level. One of these, not treated by other systems, is text searching. Figure 9 shows the animation of a two-dimensional text searching problem.

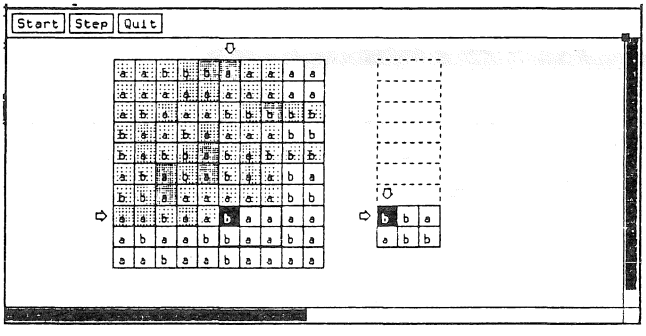


Figure 9: Text searching.

Acknowledgements

I would like to thank the people involved in the implementation of the ideas presented in this paper: Waldo Aguilar, Ivan Diaz, Edson Diaz, Luis Fuentes, Luis Jara and Gaston Quezada.

References

[BCH+85] G.P. Brown, R.T. Carling, C.F. Herot, D.A. Kramlich, and P. Souza. Program visualization: Graphical support for software development. *Computer*, pages 27–35, Aug 1985.

[BK87] J. L. Bentley and B. W. Kernighan. A systems for algorithm animation: Tutorial and user manual. Technical Report 132, AT&T Bell Labs, Murray Hill, N.J., Jan 1987.

[BMD83] Marc H. Brown, Norman Meyrowitz, and Andries Van Dam. Personal computer networks and graphical animation: Rationale and practice for education. *ACM SIGCSE Bulletin*, 15(1):296–307, Feb 1983.

[Bro88a] M. H. Brown. *Algorithm Animation.* MIT Press, 1988.

[Bro88b] M.H. Brown. Exploring algorithms using Balsa-II. *IEEE Computer*, 21(5):14–36, May 1988.

[BS84] Marc H. Brown and Robert Sedgewick. Progress report: Brown university instructional computing laboratory. *ACM SIGCSE Bulletin*, 16(1):91–101, Feb 1984.

[BS85] M. H. Brown and R. Sedgewick. Techniques for algorithm animation. *IEEE Software*, pages 28–39, Jan 1985.

[Dui86] R. A. Duisberg. Animated graphical interfaces using temporal constraints. In *Proc. ACM SIGCHI 86 Conf. Human Factors in Computing Systems*, pages 131 – 136, April 1986.

[Dui87] R. A. Duisberg. Visual programming of program visualizations. In *Proc. Conf. Visual Languages*, Aug. 1987.

[GR91] G.H. Gonnet and Baeza-Yates R. *Handbook of Algorithms and Data Structures - In Pascal and C.* Addison-Wesley, Wokingham, UK, 1991. (second edition).

[HHR89] A. Helttula, A. Hyrskykari, and K. J. Raiha. Graphical specification of algorithm animations with aladdin. In *Proc. 22nd Int'l Conf. System Sciences*, pages 892–901, 1989.

[HWF90] R. R. Henry, K.M. Whaley, and B. Forstall. The university of washington illustrating compiler. In *Proc. ACM SIGPLAN'90*, pages 223–233, White Plains, NY, June 1990.

[ISO87] S. Isoda, T. Shimomura, and Y. Ono. Vips: A visual debugger. *IEEE Software*, 4(2):8–19, May 1987.

[LD85] R.A. London and R.A. Duisberg. Animating programs using Smalltalk. *Computer*, pages 61–71, Aug 1985.

[LSV+89] T. Lehr, Z. Segall, D.F. Vrsalovic, E. Caplan, A.L. Chung, and C.E. Fineman. Visualizing performance debugging. *Computer*, 22(10), 1989.

[MR84] M.H.Brown and R.Sedgewick. A system for algorithm animation. *Computer Graphics*, pages 177–186, July 1984.

[Mye81] B.A. Myers. Incense: A system for displaying data structures. *Computer Graphics*, 17:76–93, Nov 1981.

[NVH86] J. Nievergelt, A. Ventura, and H. Hinterberger. *Interactive Computer Programs for Education*. Addison-Wesley, Reading, Mass., 1986.

[PN81] Berhard Plattner and Jurg Nievergelt. Monitoring program execution: A survey. *Computer*, 14:76–93, Nov. 1981.

[Rei83] S.P. Reiss. Pecan: A program development system that supports multiple views. Technical report, Brown University, Providence, RI, 1983.

[SGN88] Robert W. Scheifler, James Gettys, and Ron Newman. *X WINDOW SYSTEM*. Digital Press, 1988.

[Sta90] John T. Stasko. Tango: A framework and system for algorithm animation. *IEEE Computer*, 23(9):27–39, Sep 1990.

[You89] Douglas A. Young. *X Window System Programming and Applications with Xt*. Prentice Hall, Englewood Cliffs, NJ, 1989.

UN LENGUAJE IMPLICITO BASADO EN RASGOS PARA LA PROGRAMACION DE ROBOTS

Jorge E. Morales V.

Grupo de Investigación DFAC
Departamento de Ingeniería de Sistemas y Computación
Universidad de los Andes
Apartado Aéreo 4976 - Bogotá, Colombia

Resumen: En este artículo se describe un lenguaje de nivel implícito para la programación de robots en tareas de ensamblaje de piezas metalmecánicas. Es un lenguaje que utiliza, además de la información geométrica de las piezas, una descripción de sus características de ensamblaje (features), siguiendo el enfoque propuesto por [MANT90] y por [DEL91]. El proceso de programación en este lenguaje se plantea como un nuevo paradigma para la programación de robots y se aleja un poco del enfoque clásico: se parte de un borrador de solución y se va refinando, poco a poco, utilizando las herramientas de soporte del ambiente, hasta obtener una solución en términos de operaciones del robot, como lo hacen LAMA [LOZ76] y AUTOPASS [LIEB77]. La diferencia con estos lenguajes es el uso de características de los elementos involucrados en el ensamblaje, para que el proceso de refinamiento de la solución cuente con información adicional que lo guíe y poder así, restringir el espacio de búsqueda de la solución. Todo este trabajo está enmarcado dentro del desarrollo del ambiente para programación de robots MSIM [THO89] [VILLA91] [VILLE91].

Palabras Clave: Robótica, Programación de Robots, Lenguajes de Programación, Procesos de Fabricación, Modelaje por Rasgos

0. Introducción

En la actualidad, la programación de robots industriales para tareas de ensamblaje se hace a través de lenguajes explícitos, con instrucciones que describen los movimientos que debe realizar el robot para resolver un problema (tipo VAL-II [SHI84] o LM [ITM84]). También es usual que se utilice programación gestual, (aprendizaje y repetición). En los programas desarrollados con estos dos métodos, el contexto de la tarea se encuentra en la mente del programador y no se refleja en la sintaxis del programa. Además, requieren una amplia experiencia por parte del programador y se debe mantener el robot fuera de línea durante la fase de pruebas. Por estas razones, se han venido diseñando ambientes para la programación de robots donde de diferentes formas se ha tratado de apoyar la labor de programación [LOZ76] [LIEB77] [BAS89].

La arquitectura de estos ambientes va desde estructuras sencillas que manejan únicamente lenguajes de nivel explícito, hasta ambientes con herramientas de apoyo a la programación implícita, en los cuales se describe la tarea que se quiere resolver y no la forma de hacerlo. En los primeros, la programación se hace en términos de operaciones del robot; se cuenta con editores y compiladores que traducen las instrucciones de movimiento en movimientos del robot. El siguiente fragmento de código escrito en VAL-II hace que el robot se acerque a una pieza para agarrarla y muestra el tipo de instrucciones que manejan estos lenguajes:

```
move   #ppoint(30,25,25,30,45,45)
appro  pos[23],-50
departs -50
```

Los ambientes más sofisticados, por su parte, incluyen lenguajes que manejan instrucciones de más alto nivel y deben contar con un conjunto más extenso de herramientas para poder traducir dichas instrucciones en operaciones de movimiento del robot. En estos lenguajes solo se deben describir los pasos para ejecutar la tarea de ensamblaje [LOZ76][LIEB77], siendo así más fácil de entender la semántica del programa y más sencillo para el programador expresarse en dichos términos. El siguiente es un fragmento de programa en AUTOPASS para tomar una pieza e insertarla en otra:

grasp piston-pin
Insert piston-pin into piston-hole
ungrasp piston-pin

Los intentos que se han hecho para diseñar este tipo de lenguajes están basados en la descripción geométrica de los elementos activos en un proceso de ensamblaje, más un conjunto de restricciones a cumplir que expresan su estado final [LOZ76] [LIEB77] [BAS89]. Estos lenguajes no cuentan con ninguna información adicional que facilite el proceso de ensamblaje por parte del robot, lo que conlleva a algoritmos altamente combinatorios con tiempos de ejecución muy altos. Por esta razón, entre otras, su difusión a nivel comercial se ha visto muy limitada.

Este trabajo hace parte de un ambiente para la programación de robots [VILLA91] y describe un lenguaje que maneja la información geométrica de las piezas más sus rasgos o características de ensamblaje [MAN90][DEL91], que facilitan el proceso de traducción del nivel en que se expresa la solución de la tarea al nivel en el que se desarrollan las operaciones. El lenguaje propuesto está soportado por un ambiente que integra al desarrollo de programas de robots simuladores gráficos, planificadores, programación gestual, lenguajes explícitos y manejo de planes de solución. El objetivo principal de este trabajo es estudiar la factibilidad de este tipo de aproximaciones, por esto, se limita su alcance a procesos de ensamblaje de piezas metalmecánicas, ya que existen trabajos previos en modelaje de piezas por rasgos que facilitan su formalización [DEL91].

La estructura de este artículo es la siguiente: se presenta un conjunto de definiciones útiles para la compresión del tema y luego se muestra la propuesta de lenguaje de nivel implícito orientado por rasgos, incluyendo su sintaxis y su semántica.

1. Algunas Definiciones

A continuación, se presenta un conjunto de definiciones para formalizar los conceptos manejados por el lenguaje.

1.1. El Robot

Un robot industrial R es una secuencia de N cuerpos o eslabones rígidos (grados de libertad) en forma de cadena cinemática abierta simple, a través de N pares rotacionales o prismáticos. Está caracterizado por una descripción geométrica de cada

eslabón y por una descripción cinemática (parámetros de Denavit-Hartenberg [SIL89]).

1.2. El Mundo

Un objeto O del mundo está definido por su geometría (con respecto a un sistema de referencia local), por una transformación que lo ubica en el sistema de referencia del mundo y por un conjunto de rasgos del objeto, de interés en la labor de ensamblaje.

El mundo M es un conjunto de objetos, un conjunto de relaciones entre ellos (relaciones de apoyo, relaciones de encadenamiento, etc), un conjunto de relaciones entre los objetos y el robot y un sistema de referencia de base.

1.3. Una Escena del Mundo

Una escena E es el estado del modelo del mundo M en un momento dado. Un bosquejo de escena es un estado en el cual se pueden encontrar indefinidas algunas características del mundo.

1.4. Una Tarea de Ensamblaje

Una tarea de ensamblaje T es una transformación del mundo M en la cual solo se modifica la localización de los objetos, no su geometría ni sus propiedades. Está definida por dos escenas del mundo {E1,E2} que describen una situación inicial y una situación final de M.

1.5. Una Subtarea de Ensamblaje

Una subtarea de ensamblaje ST es una transformación del mundo M donde solo se modifica la localización de un objeto o algunas relaciones entre ellos expresada de forma no ambigua. Está definida por dos escenas del mundo {E1,E2}. ST puede ser cualquier transformación del mundo obtenida por una de las siguientes acciones de ensamblaje:

- agarrar
- soltar
- insertar
- mover
- localizar
- girar
- unir

1.6. Un Plan de Ensamblaje

Un plan de solución P para una tarea T especificada por {E1, E2} es una secuencia de subtareas ST1 ... STn tales que la situación inicial de ST1 es E1, la situación final de STn es E2 y la situación final de STi es igual a la situación inicial de STi+1. El programa de ensamblaje PE correspondiente a este plan está definido por la secuencia de acciones de ensamblaje A1 ... An asociadas con cada una de las subtareas.

1.7. Un Borrador de Plan

Un borrador de solución B para una tarea T especificada por {E1, E2}, es una secuencia de tareas T1 ... Tn especificadas mediante bosquejos de escena. Esto quiere decir que hay decisiones que el programador no ha concretado en su plan de solución.

1.8. Un Programa de Robot

Un programa del robot PR es una secuencia de operaciones de movimiento sobre un robot R expresada en un lenguaje explícito, (VAL-II [SHI84] o LM [ITM84]), que efectúa la tarea T definida por {E1, E2} siguiendo un plan especificado en un programa de ensamblaje P. Es decir, {E1} PR {E2}.

1.9. Un Rasgo de Ensamblaje

Un rasgo de ensamblaje RE es una característica de un objeto del mundo, útil para su manipulación por medio de un robot. Por ejemplo, los puntos de agarre de un objeto para una pinza dada, son interesantes para encontrar soluciones a un problema de ensamblaje.

1.10. Programación Implícita

Dado un modelo del mundo M, un robot R y una tarea T que describe el ensamblaje que se desea desarrollar, se debe construír un plan P de solución y un programa de robot PR que haga las transformaciones definidas en P para resolver T.

2. Descripción del Lenguaje

En esta sección se presenta una propuesta de un lenguaje implícito para la programación de robots basado en rasgis para ensamblaje mecánico.

Este lenguaje parte de las propuestas de [LOZ76] y [LIEB77] para desarrollar un lenguaje implícito, donde el programador se pueda expresar en términos de las acciones propias de tareas de ensamblaje, enriqueciendo el mundo con el concepto de rasgo para así, poder restringir el espacio de búsqueda de la solución. Adicionalmente, se incorpora el concepto de borrador de solución a partir del cual se inicia una fase de refinamiento del programa hasta llegar a una solución final.

Para ofrecer estas herramientas de desarrollo, es necesario un compilador donde el concepto de programador, como herramienta de apoyo al desarrollo de la solución, está ligado a cada uno de los procesos de la compilación. Además, este compilador cuenta con un conjunto de herramientas que se pueden agrupar según [BAS89] en:

1) Herramientas para agarrar, soltar y transportar.
Se encargan de planificar los movimientos para agarrar, soltar o transportar piezas o mover el robot en un ambiente libre de colisiones y cumpliendo con los requerimientos que se deseen (camino más corto, más seguro, rectilíneo, etc). Dentro de estas herramientas están los planificadores y el controlador para programación gestual.

2) Herramientas para construír una solución de la tarea.
Se encargan de coordinar el desarrollo del plan de solución a partir de un borrador para que no surjan conflictos entre las subtareas, por ejemplo, el haber tomado un objeto por el punto donde se debe insertar. Es importante tener en cuenta que una subtarea es altamente dependiente de las que le siguen, por lo que se debe hacer una coordinación global durante el proceso de construcción de la solución. Finalmente, estas herramientas generan, con ayuda del programador y a partir del plan de ensamblaje, un programa de robot que cumple con la tarea propuesta.

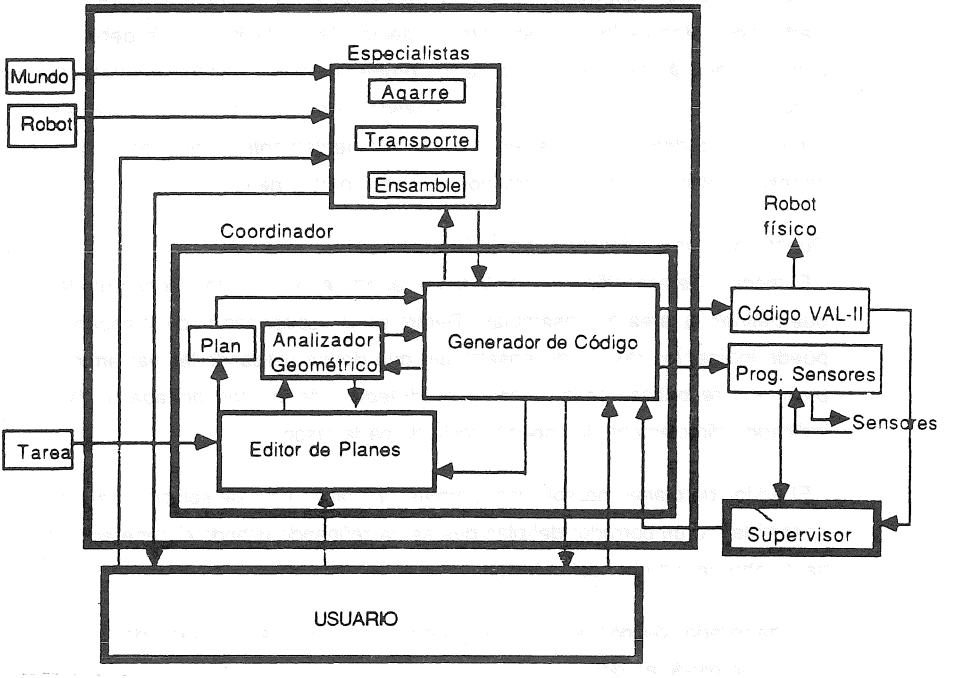
3) Herramientas para supervisar la ejecución del programa de robot.
Se encargan de coordinar el desarrollo de la ejecución, manejar información sensorial y permitir detección y corrección de errores en ejecución.

A continuación se presenta en detalle la estructura del compilador y cada uno de los módulos que lo componen. Luego, se presenta la funcionalidad del compilador y

finalmente se presenta una propuesta de sintaxis para el lenguaje de especificación de planes.

2.1. Estructura del Compilador

En esta sección se muestra la estructura operacional del compilador. Parte de la propuesta hecha en [BAS89] de un compilador compuesto por tres módulos principales: Coordinador, Supervisor y Especialistas, donde éstos últimos son submódulos para planeación y monitoreo de operaciones.



Las características de cada uno de estos módulos son:

2.1.1. El Coordinador

Trabaja desde la descripción de la tarea de ensamblaje en términos de {E1, E2}. Elabora el plan a desarrollarse basado en una descripción del borrador de la solución hecha por el programador o generada automáticamente usando una herramienta de razonamiento geométrico [PER86][LAUG87][THEV88]. Finalmente, toma el programa de ensamblaje asociado al plan y lo convierte en un programa de robot con ayuda de los especialistas y el programador.

Este módulo se encarga del control de la evolución de la solución durante la fase de traducción. Maneja los especialistas y decide las soluciones que debe adoptar del conjunto que éstos proponen para un problema dado. Cuando se detectan errores durante la planificación, decide la localidad del error para llamar al especialista respectivo o determina si el error es lo suficientemente global como para que sea necesario plantear en otros términos el plan o partes de él.

Cuenta con los siguientes submódulos principales:
- El módulo de especificación de la tarea: donde el programador construye las escenas que definen la tarea de ensamblaje. Dentro de la especificación de la escena el usuario puede indicar los rasgos de ensamblaje que desee utilizando las herramientas que le permiten reconocer rasgos como el detector de puntos de agarre [VILLE91] o indicando directamente la especificación de cada rasgo.

- El editor de planes de solución: permite definir un plan de solución para la tarea de ensamblaje o un borrador del plan que se va refinando usando el generador de código, hasta obtener un plan del solución.

- El generador de código: es el encargado de llevar la evolución de la solución. Su trabajo se divide en dos:
Cuando trabaja sobre un borrador de solución. Se encarga de guiar al programador hacia una especificación completa de cada tarea intermedia hasta obtener un plan de solución para cada una de ellas que sea compatible con el plan global. A este nivel detecta incompatibilidades y guía la replanificación de las tareas necesarias para mantener coherencia.

Cuando trabaja sobre una subtarea debidamente especificada, se encarga de traducir dicha tarea en un fragmento de programa de robot usando los especialistas e interactuando con el programador. En caso de no poder expresar la subtarea en

términos de operaciones de movimiento del robot, interactúa con el programador para replantear la subtarea en otros términos o modifcar el plan de solución desde tareas anteriores.

2.1.2. Los Especialistas

Son un conjunto de herramientas que se encargan de planificar movimientos del robot para apoyar el desarrollo de traducción de una subtarea en términos de operaciones de movimiento sobre el robot. Estos especialistas funcionan de manera semiautomática, planificando una labor específica hasta obtener una solución o fallar; en caso de presentarse una falla, consultan al programador, para que éste especifique la subtarea en otros términos o guíe la solución indicando operaciones intermedias. De ser necesario informan al generador de código la necesidad de replanificar una subtarea anterior. Hay tres especialistas en operaciones complejas:

2.1.2.1. El Especialista de Agarre

Se encarga de generar el conjunto de operaciones que permiten agarrar un objeto dado. Puede trabajar en dos modos: un modo exploratorio basado en una descripción incompleta de una acción de agarre o utilizando un punto de agarre definido a través de un rasgo. En el primer modo hace lo siguiente:

Analiza las características del estado actual del mundo y la forma como el objeto va a ser manipulado para determinar si existen puntos de agarre asociados al objeto que se puedan usar para solucionar el problema. Si la compilación se está haciendo en forma completamente automática, trata de solucionar el problema propuesto con cada punto de agarre en el conjunto. Si el programador está interactuando con el especialista, presenta el conjunto de puntos para que éste escoja el punto con el cual desea desarrollar la subtarea. Finalmente pasa a la fase de planificación de la trayectoria para el acercamiento al punto. Si no es posible solucionar el problema usando el conjunto de puntos disponibles, usa el conjunto de restricciones del mundo y el detector de puntos de agarre [VILLE91] para determinar otros posibles puntos. Luego, los puntos viables los califica por algún criterio (estabilidad, etc) y la lista se incluye en el conjunto de puntos de agarre del objeto, para repetir el proceso de selección de un punto.

El otro modo de operación es partiendo de la definición del punto de agarre por parte del usuario. Con este mecanismo evalúa la viabilidad de agarrar el objeto por dicho punto y pasa a planificar la trayectoria de acercamiento.

En planificación toma información del mundo para construír el espacio de búsqueda y planear los movimientos para llegar hasta el punto de agarre usando un planificador de movimientos finos [VILLE91]. El resultado es una solución al problema de agarre para que el coordinador a través del generador de código o el programador, escoja si es la que le conviene para su tarea global. En el código de nivel explícito generado para desarrollar la operación incluye un esquema de retroalimentación basado en información sensorial obtenida durante la ejecución.

En caso de detectar un error durante esta planificación (no se puede agarrar por ese punto porque está obstruído, por ejemplo), retorna a la fase de selección de un punto en modo exploratorio.

Finalmente durante la ejecución, el especialista está listo para corregir errores locales haciendo una replanificación de la trayectoria o detectar otros puntos de agarre (puede entrar en modo de operación automática durante la replanificación).

2.1.2.2. El Especialista de transporte

Este especialista se encarga de generar trayectorias libres de colisión para llevar el manipulador hasta los puntos de agarre o de ensamble, con o sin objeto en la pinza. Esta labor la hace usando una representación discreta del espacio de configuraciones del manipulador [LOZ83][LOZ87][MOR91] para los movimientos gruesos y un planificador por potenciales [VILLE91] para movimientos finos. En caso de no encontrar solución a un problema de planificación, interactúa con el programador para que éste indique puntos intermedios que faciliten la planificación automática o que explícitamente indique la ruta que se debe seguir para cumplir la trayectoria.

En tiempo de ejecución está pendiente de replanificar para corregir errores generando otras posibles soluciones.

2.1.2.3. El Especialista de Ensamble

Se encarga de ensamblar las piezas en la posición final requerida. Puede operar en dos modos: Un modo exploratorio, donde busca las posibles formas de ensamblar los elementos, usando una herramienta que hace razonamiento geométrico para determinar una forma de ensamblaje entre dos elementos definidos por una o varias piezas [PER86][LAUG87][THEV88]. Pasa luego a la fase de planificación de movimientos usando un planificador de movimientos finos [VILLE91]. El otro modo de operación parte de las superficies de ensamble definidas por el programador y a partir de allí planifica la trayectoria a seguir para ensamblar los elementos.

Al planificar desarrolla una serie de alternativas para usar en caso de detección de errores. De no encontrar una alternativa viable, el programador puede sugerir soluciones o indica explícitamente la forma de hacer el ensamblaje. De no existir solución, informa al generador de código para el replanteamiento del plan. Además, utiliza la herramienta de razonamiento geométrico para determinar evoluciones en la escena, tales como nuevas relaciones de encadenamiento de objetos, apoyo, etc.

Este especialista incluye puntos de chequeo sensorial dentro del código generado para que en ejecución se tomen alternativas o se replanifique sobre errores de ser necesario.

2.1.3. El Supervisor

El supervisor es el encargado de llevar el control de ejecución del programa del robot en tiempo real. Recibe toda la información disponible en los sensores, la analiza y determina la mejor respuesta para un requerimiento dado. Se encarga de mantener actualizado el modelo del mundo para poder replanificar a partir de cualquier punto de la solución. En caso de detección de errores, se encarga de interrumpir la ejecución de la tarea y darle la información necesaria al controlador para que replanifique a partir del punto donde se detectó el error.

2.2. Funcionalidad del Compilador

El compilador globalmente se puede visualizar como dos grandes módulos: primero el especificador, encargado de la generación de un borrador del plan de solución y segundo el generador de código explícito, encargado de refinar e interpretar dicho plan hasta obtener una solución a la tarea especificada en términos de operaciones del robot.

2.2.1. Especificación del Plan de Solución

Se hace a través de los submódulos del módulo Controlador del compilador. Usa el lenguaje para especificar borradores de solución al problema de ensamblaje o para especificar un plan de solución completo. Para especificar un borrador, el programador puede dejar sin especificar elementos de las instrucciones, por ejemplo GRASP OBJ1, donde no se especifica el punto de agarre por el cual se debe cumplir la labor, de tal forma que no está indicando completamente la transformación del mundo. También puede omitir operaciones, bien para que el compilador las genere automáticamente o para especificarlas durante la traducción, por ejemplo INSERT OBJ1 INTO OBJ2, donde no se especifica la operación de agarrar el OBJ1.

Para obtener el borrador del plan general de solución, el programador puede hacer uso de la herramienta de razonamiento geométrico [PER86][LAUG87][THEV88] para que se genere un esqueleto del plan de solución y refinarlo durante la interpretación; de esta forma, es posible generar varios planes a partir de un borrador inicial, para que se puedan analizar las ventajas de unos y otros antes de escoger la solución que se desea utilizar.

Para obtener el programa final del robot el programador puede hacer evolucionar el borrador de tres formas diferentes:
- trabajar secuencialmente en la obtención de la solución. Es decir, para cada una de las tareas que definen el borrador, especificarla hasta poder obtenerla en términos del lenguaje explícito y solo al finalizarla pasar a la siguiente tarea.

- refinar el borrador iterativamente hasta obtener el plan de solución y solo después de completarlo empezar la traducción de cada subtarea en código explícito.

- combinando los dos anteriores, de tal forma que inicialmente compila las tareas bien definidas y con base en éstas refina la especificación de las demás hasta completar la solución.

2.2.2. Interpretación de un Plan de Solución

Para pasar a la fase de interpretación de un plan de solución no es necesario que el plan esté completamente especificado, sino que se puede partir de un borrador y refinarlo. El generador de código cuenta con los especialistas para traducir las subtareas en operaciones del lenguaje de nivel explícito porque, en general, cada acción de ensamblaje corresponde a un conjunto de operaciones sobre el robot [LOZ76]. Posee un conjunto de herramientas para la labor de traducción, adicionales a las que ofrecen los especialistas, como son:
- Un planificador de tareas sencillas para completar un borrador con las subtareas omitidas por el programador. Este planificador es bastante simple. Chequea los prerrequisitos de cada subtarea e incluye antes las acciones que se omitieron para que el intérprete las complete según el modo de operación que se esté usando.

- Herramientas para programación gestual: modo *teach*, movimiento de las articulaciones, movimientos en coordenadas del mundo, de la pinza, etc.

- Simulador interactivo de las operaciones sobre el robot.

Para iniciar la traducción de una subtarea, primero se completa la especificación en forma automática o según especificación del programador. Es decir, si el programador

Indica el formato completo de la subtarea, se pasa a la fase de generación de código; pero si hay omisiones en la especificación, se usan los especialistas, de tal forma que no se hace una planeación particular para una subtarea muy específica, sino que se exploran todas las posibles soluciones a la subtarea y se escoge alguna en forma automática o por decisión con el programador.

En caso de que falten subtareas prerrequisito de la tarea, por ejemplo una operación de agarre para desarrollar una operación de ensamble, se usa el planificador de tareas sencillas para especificarlas y se interpretan antes de la subtarea actual.

La labor de interpretación de una subtarea se puede hacer de tres modos diferentes dependiendo de lo que desee el programador:

1) **Modo Automático**: El programador especifica al intérprete una instrucción dada. Este cuenta con un formato de traducción de cada subtarea (instrucción) en un conjunto de operaciones del lenguaje explícito más las pre y postcondiciones de cada una de ellas [LIEB77]. Si el estado del mundo permite la ejecución de la acción entonces entra a tratar de traducir dicha acción en un conjunto de operaciones expresadas en el lenguaje explícito, haciendo uso del formato especificado para la subtarea. En caso de no encontrar una solución posible a la acción especificada, entonces interactúa con el programador para que este guíe la solución indicando pasos intermedios, replanteando la acción de otra forma o pasando a otro modo de operación del intérprete y tomar parte activa en la solución de la subtarea. Esta replanificación por parte del programador puede implicar modificar o replanificar subtareas anteriormente interpretadas.

2) **Modo Explícito:** El programador, en este modo, toma el control de la traducción y especifica directamente un conjunto de operaciones en el lenguaje explícito que cumplen con la acción deseada.

3) **Modo Gestual:** El programador toma control directo del modelo del robot en el ambiente y lo manipula modificando los valores articulares. En este modo cuenta con la posibilidad de enseñarle posiciones al manipulador y grabarlas, incluyéndolas en la solución.

Durante el proceso de interpretación de una acción el programador puede hacer uso de las herramientas de que dispone en cualquiera de los tres modos como son especialistas, planificadores, modo teach, etc. y enriquecer las definiciones en los objetos (rasgos). Independientemente del modo bajo el cual el programador esté operando, siempre tiene una concepción global de los elementos del modelo como son

objetos, manipulador y rasgos e igualmente puede referirse a ellos a lo largo de la compilación desde cualquier nivel (plan, explícito o gestual).

2.3. Lenguaje de Especificación de la tarea

El lenguaje de Especificación de la tarea está dividido en dos: La parte declarativa que especifica las características de la escena y una parte operativa que describe el conjunto de acciones necesarias para desarrollar la tarea.

Globalmente, para definir el lenguaje de especificación de la tarea, se tomó en cuenta que la sintaxis del lenguaje puede ser sujeta a cambios para incluír el lenguaje dentro de un lenguaje informático ya existente, de tal forma que se puedan usar las facilidades que éste ofrece como son control de flujo, declaración de variables auxiliares, etc.

A continuación se presenta el conjunto de instrucciones básicas para la especificación del plan de solución:

INSERT Obj1.<nombre rasgo SE> **INTO** Obj2.<nombre rasgo SE> /

INSERT Obj1.<nombre rasgo SE> **IN** Obj2.<nombre rasgo SE> /

GRASP Obj.<nombre rasgo PA> /

FREE Obj /

PLACE Obj1.<nombre rasgo SS> **ON** Obj2.<nombre rasgo SS> /

FASTEN Obj1 **WITH** Obj2 /

UNFASTEN Obj1 **FROM** Obj2 /

NAME Ob1,...,Objn **ASSEMBLY** Objm /

TURN Obj **AROUND** <EJE> <numero de grados> /

MOVETO <PUNTO> /

MOVETO <TRANSFORMADA>

2.4. Tipos de Rasgos

Debido a que la cantidad de rasgos depende del alcance que se pretenda dar al sistema y de la variedad de objetos que se le permitirán ensamblar, definir el conjunto de

rasgos de un sistema es un problema para el cual no existe una metodología [MAN90]. Por lo tanto, la especificación de rasgos que se hace a continuación parte del conocimiento adquirido en el área de ensamblaje y diseño por rasgos [DEL91]. Se especifican tres tipos de rasgos que se seleccionaron tomando en cuenta la estructura definida para el compilador. Es decir, están involucrados en cualquier tipo de ensamblaje mecánico y se reconocen a partir del conjunto de acciones que el compilador debe realizar para interpretar una subtarea.

2.4.1. Rasgo: Punto de Agarre (PA)

2.4.1.1. Descripción:

Un punto de agarre se define como un punto asociado a un objeto, tal que por ese punto el objeto puede ser agarrado por un robot cumpliendo con los requerimientos de estabilidad necesarios. Es importante indicar que son dependientes no solo del objeto sino también del tipo de pinza con que cuente el robot para tomar el objeto.

2.4.1.2. Caracterización:

Un punto de agarre se especifica con:
- Una transformada: Especifica la transformación que define la posición en la cual debe quedar el sistema de referencia de la pinza sobre el objeto.

- Una distancia: Especifica un punto situado a dicha distancia sobre el eje Z del sistema de referencia de la pinza. A partir de este punto se debe desarrollar la planificación de movimientos para GRASP.

- Una apertura: Especifica la apertura de entrada para acercamiento.

2.4.2. Rasgo: Sección de Ensamblaje (SE)

2.4.2.1. Descripción:

Una sección de ensamblaje es un conjunto de elementos geométricos que pertenecen a un mismo objeto y que corresponden a una sección que entra en contacto (se ensambla) con otro objeto durante el proceso de ensamblaje.

2.4.2.2. Caracterización:

Una sección de ensamblaje se caracteriza globalmente por:

- Una transformada: Especifica la transformación que define la posición en la cual debe quedar el sistema de referencia de la otra sección de ensamblaje involucrada en el ensamble.

- Un conjunto de objetos: define el conjunto de objetos con los que se puede ensamblar dicho objeto por esa sección de ensamblaje.

2.4.3. Rasgo Sección de Soporte (SS)

2.4.3.1. Descripción:

Una sección de soporte se define como conjunto de elementos geométricos que componen una superficie de un objeto, sobre los cuales se puede apoyar el objeto en forma estable para colocarlo sobre una superficie base. La misión principal de estas secciones de soporte es apoyar las labores de ensamblaje donde sean necesarios pasos intermedios para tomarlo de otra forma.

2.4.3.2. Caracterización:

Una SS se caracteriza por:
- Una transformada: Especifica la transformción que define la posición en la cual debe quedar el sistema de referencia de la superficie que se vaya a usar como base.

4. Conclusiones

En este artículo se presenta un lenguaje de nivel implícito orientado por rasgos que pretende apoyar la programación de robots para tareas de ensamblaje. Este lenguaje parte de un nuevo paradigma de programación propuesto en [VILLA91] y que pretende integrar al lenguaje características del ambiente para facilitar con esto la programación de robots, ofreciendo la mayor cantidad de herramientas disponibles para dicha labor.

Es importante aclarar que el uso que se le de a una herramienta de estas características está sujeto a una metodología a partir del paradigma propuesto y cuya especificación debe ser el siguiente paso en la búsqueda de una nueva forma de programación de robots.

Agradecimientos

El trabajo que se presenta en este artículo es producto del esfuerzo conjunto de un grupo de trabajo en el área de robótica en la facultad de Ingeniería de la Universidad

de Los Andes, del cual se debe destacar al profesor Jorge Villalobos S. y al ingeniero Eduardo Villegas J.

Bibliografía

[BAS89] Basañez, L., Operation Specialists for Automatic Programming and Monitoring of Robotic Assembly Cells, Journal of Robotics and Computer Integrated Manufacturing, 1989.

[DEL91] De la Rosa, F., Una Propuesta de Diseño Asistido por Computador de Piezas Metalmecánicas Representados por Rasgos, Universidad de Los Andes, Tesis de Magister, Junio 1991.

[ITM84] ITMI (Industrial Technology and Machine Intelligence), LM Reference Manual, 1984.

[LAUG87] Laugier, C., Raisonnement Geometrique et Methodes de Decision en Robotique. Aplication a la Programmation Automatique des Robots, Tesis de doctorado, Institut National Polytechnique de Granoble, diciembre 1987.

[LIEB77] Lieberman, L., AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly, IBM Journal of Research and Development, Vol. 21, No. 4, Julio 1977.

[LOZ] Lozano-Perez, T., Lama: A language for Automatic Mechanical Assembly, MIT.

[LOZ76] Lozano-Perez, T., The design of a Mechanical Assembly System, MIT, 1976.

[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 Manipulators, IEEE Journal of Robotics and Automation, Vol. RA-3, No. 3, junio 1987.

[MANT90] Manthila, M., A Modeling System for Top-Down Design of Assembled Products, IBM Journal Res. Develop, Vol. 34, No. 5, septiembre 1990.

[MOR91] Morales, J., Planificador de Trayectorias basado en Espacios de Configuraciones Discretizados, Universidad de Los Andes, Informe Interno, 1991.

[PER86] Pertin, J., Modelisation du Raisonnement Geometrique pour la Programmation des Robots, tesis de Doctorado, Institut National Polytechnique de Grenoble, 1986.

[SHI84] Shimano, BI, VAL-II: A New Robot Control System for Automatic Manufacturing, Proc. of the Int. Conference on Robotics, marzo 1984.

[SIL89] Silva M., Introducción a la Robótica Industrial, Universidad de Zaragoza, Mayo 1989.

[THEV88] 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 Programación Fuera-de-línea y Simulación de un Robot PUMA 560, Instituto de Cibernética, Barcelona, 1989.

[VILLA91] Villalobos, J., MSIM: Un Ambiente de Programación de Robots con Capacidad de Planificación de Tareas, Universidad de Los Andes, 1991.

[VILLE91] Villegas, E., Algoritmos Eficientes para Resolver los Problemas de Visualización, Colisión y Planificación en un Ambiente de Programación de Robots, Universidad de Los Andes, 1991.

Un Experimento en Aprendizaje Automático y Evaluación Parcial

David A. Fuller †
Sacha Bocic

Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Casilla 6177
Santiago 22
Chile

Resúmen: En este trabajo se presenta un experimento en aprendizaje automático, y específicamente, en generalización basada en ejemplos. Primero, se presenta una introducción al tema de aprendizaje y al de generalización basada en ejemplos. Además, se explican los algoritmos conocidos y sus problemas. Luego, se presentan varias extensiones a dicho algoritmos con el fin de permitir el aprendizaje de ejemplos de gran tamaño. Finalmente, se prueba el nuevo algoritmo con un ejemplo del campo de los lenguajes de programación, y se entregan conclusiones y trabajo futuro.

† E-mail: dfuller@{dcc.puc.cl, pucing.bitnet}, fax: +(56-2) 552-4054.

1. Introducción

Todo sistema de aprendizaje realiza una transformación de la información provista por un "tutor" (o el medio ambiente) de tal forma de almacenarla para su posterior uso. El grado de complejidad en la transformación de la información determina el nivel de complejidad de la técnica. Así, a mayor grado de complejidad, menor es el esfuerzo que debe realizar el tutor y mayor el esfuerzo del sistema.

La técnica que menor grado de transformación de la información involucra es el "aprender de memoria", donde el sistema sólo se limita a aceptar y almacenar lo que el tutor indica. Otra técnica es la conocida como "aprender por instrucción", en la cual se hace alguna transformación de la información pero sólo a nivel sintáctico. En "aprendizaje por deducción", el sistema infiere a partir del conocimiento, conclusiones útiles las cuales almacena para el futuro. Estas deducciones tienen la característica de ser preservadoras de la verdad. En otras palabras, de un conjunto de reglas lógicamente verdaderas se deducen hechos que también son verdaderos. En "aprendizaje por analogía" tenemos una combinación de inferencia deductiva e inferencia inductiva. En esta técnica, descripciones de distintos dominios son comparados para determinar una estructura común, la cual sirve como base para transformar valores desde un dominio al otro. La determinación de la estructura común involucra un proceso de inferencia, y la transformación de un dominio al otro involucra un proceso de deducción. "Aprendizaje inductivo" es la técnica que más transformación de la información involucra y por lo tanto, es la técnica cuya implementación presenta mayor grado de dificultad. En esta técnica, la inferencia es inductiva, lo que quiere decir que la inferencia comienza con un conjunto de hechos y se produce una generalización de estos hechos. Este tipo de inferencia (al contrario de la inferencia deductiva) no es preservadora de la verdad, pues es posible inferir una regla falsa a partir de un conjunto de reglas cuyo valor de verdad es verdadero. Por ejemplo, si sabemos que mi gato tiene el pelo color negro, no es válido deducir que todos los gatos tienen el pelo color negro.

El aprendizaje inductivo se divide en dos sub-áreas, "aprendizaje basado en ejemplos" y "aprendizaje por observación y descubrimiento". En aprendizaje basado en ejemplos, el objetivo consiste en construir una generalización capaz de aceptar todos los ejemplos positivos de algún concepto objetivo y de rechazar todos los ejemplos negativos. Existe aprendizaje basado en múltiples ejemplos y

aprendizaje basado en un solo ejemplo. Por último, en aprendizaje por observación y descubrimiento, el sistema busca patrones o reglas que le permitan explicar todas o la mayoría, de las observaciones que éste hace. Este proceso se realiza sin la guia de un tutor.

Teniendo esto en cuenta, nos concentraremos en el estudio de una de las técnicas de aprendizaje conocida como "Generalización Basada en Ejemplos", a la cual nos refereriremos por su sigla en inglés **EBG** (Explanation Based Generalization), la que se encuentra en el área de aprendizaje inductivo. En este artículo, se presentarán algunas experiencias en aprendizaje basado en ejemplos, lo que llevará a proponer un nuevo algoritmo. Se demostrará la potencialidad de este nuevo algoritmo con un ejemplo en el área de lenguajes de programación. El sistema será capaz de aprender un lenguaje imperativo y procesar programas en dicho lenguaje. Hasta la fecha, los algoritmos tradicionales no son capaces de procesar ejemplos de esa complejidad. Además, se hará la conexión cada vez más evidente entre EBG y evaluación parcial, esta última, una técnica de evaluación simbólica [Fuller 89].

2. Generalización Basada en Ejemplos

El término EBG es mencionado por primera vez en [Mitchell 86]. Esta técnica ataca el problema de formular conceptos generales a partir de un solo ejemplo. El algoritmo que utiliza EBG consta de dos etapas. En la primera, se analiza el ejemplo de entrenamiento y se construye una explicación de porqué el ejemplo de entrenamiento es una instancia del concepto a generalizar. La segunda etapa consiste en generalizar la explicación para obtener una reformulación del concepto.

La explicación que construye EBG forma un árbol de prueba [Lloyd 84]. Antes de continuar, es necesario definir cuatro conceptos que son importantes en todo sistema EBG .

Definición 2.1: Concepto objetivo corresponde a una definición no operacional del concepto a aprender. ❑

Definición 2.2: Criterio operacional corresponde a la especificación del tipo de definiciones de predicados que se consideran operacionales. En [Mitchell 86], se

representa este criterio como una lista de predicados observables (hechos) o fácilmente calculables. Otra forma de especificar los predicados operacionales consiste en especificar en una lista, los predicados que se consideran no operacionales. Una descripción de un predicado se dice que es operacional si y sólo si es expresada en términos de predicados operacionales. ❑

Definición 2.3: Un ejemplo de entrenamiento corresponde a la descripción de un objeto, el cual es una instancia del concepto objetivo. ❑

Definición 2.4: Teoría de dominio corresponde a un conjunto de reglas que definen el dominio desde el cual el ejemplo de entrenamiento y el concepto objetivo pueden ser derivados. Las reglas deben ser capaces de demostrar que el ejemplo de entrenamiento es una instancia del concepto objetivo. ❑

Como se mencionó, el algoritmo consta de dos etapas, la primera que es la construcción del árbol de prueba y la segunda que consiste en la generalización, tomando como base el árbol de prueba.

La construcción del árbol de prueba se hace con el ejemplo de entrenamiento en la raiz y la construcción de los nodos se hace utilizando las reglas que conforman la teoría de dominio, el cual no necesariamente es un hecho. La expansión (o ramificación) del árbol de prueba se termina al llegar a un predicado operacional. Para determinar si un predicado es o no operacional, se utiliza el criterio operacional. Si para un ejemplo de entrenamiento no es posible construir un árbol de prueba, entonces el ejemplo de entrenamiento no es válido, ya que no es posible demostrar su validez utilizando la teoría de dominio y por lo tanto no es posible generalizar.

La generalización consiste en determinar un conjunto de condiciones suficientes bajo las cuales la explicación se sustenta en términos de predicados que cumplan con el criterio operacional. Esto, en términos del árbol de prueba, consiste en la conjunción de los nodos operacionales (hojas del árbol) pero con variables frescas. Este proceso es explicado en [Kedar-Cabelli y McCarthy 87] como la forma en que se propagan las sustituciones de la reglas, pero ignorando las sustituciones de los hechos cuando se crea el árbol de prueba.

Esto nos hace pensar que si bien es cierto que las dos etapas del algoritmo son independientes, las podríamos realizar simultaneamente. Es decir, podríamos construir el árbol de prueba y al mismo tiempo podríamos ir construyendo nuestra generalización, pero teniendo cuidado de no instanciar variables en los predicados operacionales.

Supongamos que utilizamos el conjunto de reglas de la Figura 2.5 como una teoría de dominio, el predicado `puedecomprar(juan,ferrari)` como un ejemplo de entrenamiento y buscamos la generalización de `puedecomprar(X,Y)` considerando como predicados operacionales a los hechos (`esauto`, `sevende`, `gusta`, `costo`, `ahorro` y `prestamo`).

```
puedecomprar(Persona, Auto) :-
                    gusta(Persona,Auto),
                    sevende(Auto),
                    pagable(Persona, Auto).
pagable(Persona, Auto) :-
                    costo(Auto, X),
                    ahorro(Persona, Xa),
                    Xa > X.
pagable(Persona, Auto) :-
                    costo(Auto, X),
                    ahorro(Persona, Xa),
                    Xa < X,
                    prestamo(Persona, X-Xa).

esauto(ferrari).
sevende(ferrari).
gusta(juan, ferrari)
costo(ferrari,300000).
ahorro(juan,0).
prestamo(juan,300000).
```

Figura 2.5: Ejemplo con teoría de dominio.

Así,en una primera etapa, EBG comienza a construir el árbol de prueba para el ejemplo de entrenamiento. Simultáneamente, construimos una "copia" del árbol de prueba, pero con variables sin instanciar (es decir, variables libres). A esta copia le llamaremos árbol generalizado. Inicialmente, se crea el árbol de prueba y el generalizado con un sólo nodo, tal como lo muestra la Figura 2.6.

Figura 2.6: Arboles parcial y generalizado al iniciar el proceso.

Utilizando la teoría de dominio, es posible expandir los árboles de prueba y generalizado, como se muestra en la Figura 2.7, donde se ha utilizado para dicho efecto una representación de árboles AND/OR [Bratko 90].

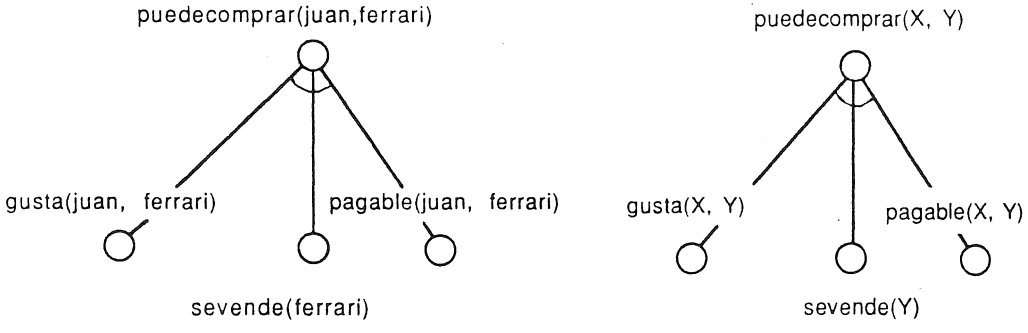


Figura 2.7: Arboles de prueba y generalizado durante el proceso.

Los predicados `gusta(juan,ferrari)` y `sevende(ferrari)` son considerados predicados operacionales, puesto que son hechos. Como sabemos, la expansión de una rama del árbol termina al llegar a un nodo o predicado operacional, por lo que sólo nos queda por expandir el nodo `pagable(juan, ferrari)`. La expansión de este nodo nos lleva al árbol final, ya que todos los nodos de las hojas son operacionales. El árbol generalizado final se muestra en la Figura 2.8 (sólo se grafican las expansiones exitosas).

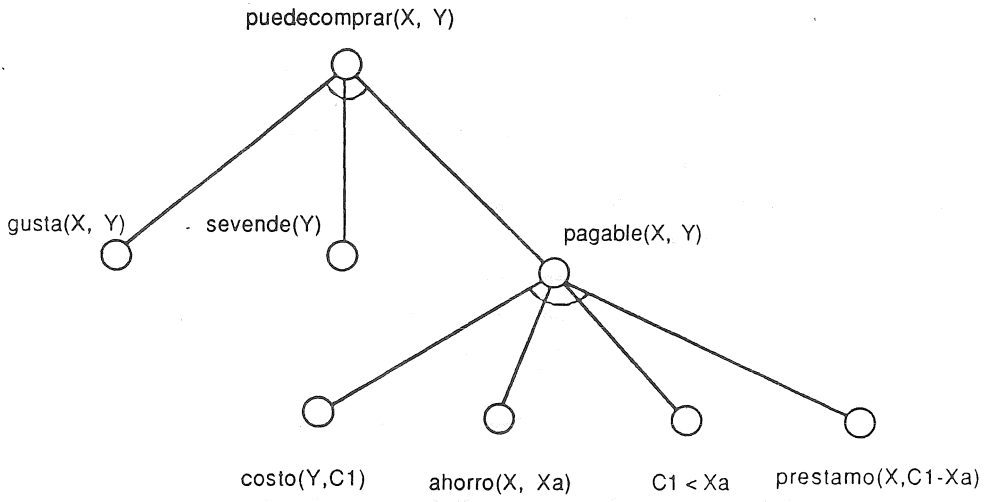


Figura 2.8: Arbol generalizado al final del proceso.

Una vez que se termina de construir el árbol de prueba, estamos en condiciones de generar la reformulación del concepto objetivo (`puedecomprar(X, Y)`) en términos de los predicados operacionales. Esto se hace formando la conjunción de los predicados operacionales del árbol generalizado, obteniendo el siguiente resultado:

Reformulación = `gusta(X,Y), sevende(Y), costo(Y, C1),`
`ahorro(X, Xa), Xa<C1, prestamo(X, C1-Xa)`

Resumiendo, hemos obtenido una reformulación del concepto objetivo en términos de predicados operacionales, guiados por el ejemplo de entrenamiento.

3. Implementación de EBG1

A continuación se presenta una implementación del algoritmo EBG tomada de [van Harmelen y Bundy 88] que le llamaremos **EBG1**.

```
ebg1( Hoja, HojaGen, HojaGen) :-
        operacional(Hoja), ! ,
        call(Hoja).[1]
ebg1( (Goal1,Goal2), (GenGoal1,GenGoal2), (Hojas1,Hojas2) ) :-
         ebg1(Goal1, GenGoal1, Hojas1),
         ebg1(Goal2, GenGoal2, Hojas2).
ebg1( Goal, GenGoal, Hojas) :-
         clause(GenGoal, GenClause),[2]
         copy( (GenGoal:- GenClause), (Goal:-Clause) ),[3]
         ebg1( Clause, GenClause, Hojas).
```

Figura 3.1: Implementación de `ebg1`.

El predicado `ebg1` toma como primer argumento un ejemplo de entrenamiento, el segundo argumento corresponde al concepto objetivo y entrega en el tercer argumento la reformulación del concepto objetivo en términos de predicados operacionales. En este algoritmo, la teoría de dominio y el criterio operacional se encuentran como entrada en forma implícita, ya que están en la base de datos de PROLOG.

Ejemplo 3.2: El primer ejemplo que analizaremos corresponde a un *clásico* de la literatura de aprendizaje y lo usaremos como entrada para la implementación de EBG1. Consideremos la siguiente base de datos en PROLOG:

```
% Concepto Objetivo
apilar (X,Y) :- mas_liviano(X,Y).
% Ejemplo de Entrenamiento
esta_sobre(caja1,mesa1).
volumen(caja1,10).
es_un(caja1,caja).
es_un(mesa1,mesa).
```

---

1. `call(Hoja)` es equivalente a un llamado al predicado que representa la variable `Hoja`.

2. `clause(GenGoal,GenClause)` es verdadero si existe en la base de datos de Prolog una definición de la forma `GenGoal :- GenClause`.

3. El predicado `copy` se usa para que el árbol generalizado sea una copia exacta del árbol de prueba pero con variables frescas.

```
color(mesa1,azul).
color(caja1,rojo).
densidad(caja1,10).
% Teoría de Dominio
mas_liviano(X,Y) :- peso(X,W1),
peso(Y,W2),
menor(W1,W2).
peso(X,500) :- es_un(X,mesa).
peso(X,Y)  :- volumen(X,V),
densidad(X,D),
multiplica(V,D,Y).
% Criterio Operacional
operacional(Goal) :- member(Goal, [multiplica(_,_,_), menor(_,_)
                                   esta_sobre(_,_), volumen(_,
                                   es_un(_,_), color(_,_),
                                   densidad(_,_)] ).
```

Figura 3.3: Ejemplo clásico.

El predicado `menor(X,Y)` es cierto si el valor de la variable `X` es menor que el valor de la variable `Y`. En forma análoga, el predicado `multiplica(X,Y,Z)` es verdadero si el valor de la variable `Z` es igual al producto de los valores de las variables `X` e `Y`. Estos predicados los consideramos predefinidos por el lenguaje.

Así, para generalizar el predicado `apilar` usamos la implementación de EBG1 descrita antes, de la siguiente forma:

```
ebg1(apilar(caja1,mesa1), apilar(X,Y), Reformulacion).
```

donde `apilar(caja1,mesa1)` corresponde al ejemplo de entrenamiento, `apilar(X,Y)` es el concepto objetivo y `Reformulacion` es un variable inicialmente desinstanciada que entregará la reformulación del concepto objetivo, es

4. El predicado `member` recibe como primer argumento un elemento y como segundo una lista. Si el elemento pertenece a la lista, entonces el predicado es verdadero.

decir, una nueva definición del predicado `apilar` en función de los predicados operacionales. El resultado obtenido es:

```
Reformulacion =  volumen(X,Vx), densidad(X,Dx), multiplica(Vx,Dx,Mx),
                 es_un(Y,mesa),  menor(Mx,500).
```

Es decir, cualquier cosa de menos de 500 kilos se puede poner sobre una mesa. Nótese que en la reformulación, todos los predicados tienen variables sin instanciar salvo el predicado `menor`, que tiene una constante (500). Esta constante proviene de la instanciación que se produce al expandir el árbol de prueba y el árbol generalizado según la primera definición del predicado `peso`. Como este predicado no es operacional, la variable se instancia.

4. Extensiones a EBG

En el Ejemplo 3.2, los predicados operacionales eran casi todos hechos ("facts"), salvo los predicados `multiplica` y `menor` que son predicados predefinidos por el lenguaje. Además, el ejemplo de entrenamiento contenía suficiente información como para "guiar" al algoritmo por el espacio de búsqueda, de tal forma de encontrar una solución única. Estos dos hechos no son siempre verdaderos, especialmente cuando se trabaja con ejemplos de mayor tamaño. A continuación, se propondrán dos extensiones al algoritmo EBG1, las que se incorporarán en una nueva versión del algoritmo EBG.

El algoritmo EBG1 expande cada rama del árbol de prueba hasta llegar a un predicado operacional. Como se sabe, en este punto no se producen instanciasiones en las variables del árbol generalizado y el efecto en la reformulación del predicado original se limita a la incorporación de éste a dicha reformulación. Gráficamente:

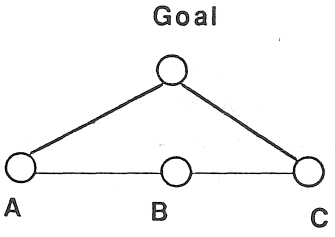


Figura 4.1: Primer problema de EBG1.

Supóngase que para el árbol de prueba de la figura, los predicados `A`, `B` y `C` son predicados operacionales. En consecuencia, la reformulación del predicado `Goal` es la conjunción de los predicados operacionales de dicho árbol, de la forma `Goal :- A, B, C.`

Ahora supongamos que no se desea que este predicado `B` aparezca en la reformulación del concepto objetivo. Si es este el caso, se tiene que eliminar al predicado `B` de los predicados operacionales. Al hacer esto, el algoritmo tendrá que expandir totalmente este predicado y se va a llegar a una conjunción de predicados que corresponde a la generalización de este predicado `B`.

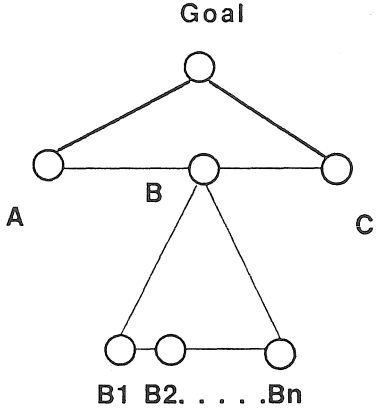


Figura 4.2: Expansión de `B`.

De esta forma nos encontramos con que la solución es peor que el problema, pues si no queremos al predicado `B`, menos queremos a la generalización de

este predicado[5]. Además, no se puede optar por sacar a este predicado de la reformulación del concepto objetivo pues se estaría generando una reformulación inválida. En efecto, los predicados reciben variables como argumento, esto implica que, o se instancian variables o se verifica que se cumpla alguna condición. Si se elimina a este predicado "no deseado" estamos eliminando estas instanciaciones o verificaciones de condiciones y por lo tanto la reformulación no será válida.

Una forma de superar este problema consiste en reemplazar la expansión del árbol de prueba y del árbol generalizado por un meta-llamado a este predicado y su respectiva copia. De esta forma, todas las variables que se tienen que instanciar en este predicado lo hacen, así como también se verifican todas las condiciones, pero no se incluye este predicado no deseado en la reformulación. Para ello, el usuario debe definir los predicados que son posibles de ejecutar y para que el algoritmo modificado los ejecute.

Es posible especificar los predicados "ejecutables" de una forma similar a la que se usa para especificar los predicados operacionales, es decir, mediante una lista y a través del predicado `member` se detecta si el predicado pertenece a dicha lista. La forma de incorporar esta idea al algoritmo primitivo es mediante la extensión de la definición original del predicado `ebg1` y la definición del predicado `ejecutable`, de la siguiente forma:

```
ebg(Goal,GenGoal,[]) :-
                    ejecutable(Goal),
                    call(Goal),
                    call(GenGoal),!.

ejecutable(Goal) :- member(Goal, [lista de predicados ejecutables]).
```

Figura 4.3: Incorporación de la primera extensión.

En el Ejemplo 3.2, se vió que el ejemplo de entrenamiento nos guiaba por el espacio de búsqueda de tal forma de llevarnos a una única reformulación del concepto objetivo. En efecto, en este ejemplo para el predicado `peso` tenemos

5. Si el predicado `B` es un hecho, no hay forma de eliminarlo de la reformulación.

dos definiciones, lo que en términos de un árbol AND/OR constituye un nodo-or, sin embargo, el algoritmo escoge sólo una de ellas.

Este es precisamente el rol del ejemplo de entrenamiento, guiar al algoritmo a través del árbol de prueba, escogiendo algunos nodos para expandir y descartando otros. ¿Pero cómo enfrentamos el problema de no saber cuál nodo expandir si el ejemplo de entrenamiento no tiene la suficiente información? Una posible solución consiste en modificar el algoritmo EBG para que detenga la expansión de los árboles al encontrarse con nodos-or en los cuales se presenta este problema. Esta solución presenta el problema que en muchos casos la terminación anticipada de la expansión de los árboles nos lleva a obtener árboles muy poco profundos, con lo que la reformulación del concepto objetivo no presenta un gran avance.

La otra solución consiste en expandir cada una de las soluciones, de tal forma de obtener tantas reformulaciones del concepto objetivo como expansiones del árbol de prueba se realicen, pero ¿podemos estar seguros de que el número de expansiones (y reformulaciones) será finito? Ciertamente no, y como prueba de ello analicemos el siguiente ejemplo, en el que se desea reformular el predicado `largolista(Lista,N)`, cuya definición es la siguiente:

```
largolista([] , 0).[6]
largolista([Cabeza | Cola] , N) :-
                    largolista(Cola, Nc),
                    N is Nc + 1.
```

Figura 4.4: Definición de `largolista`.

Una de las formas de usar este predicado es recibiendo como primer parámetro una lista y entregando en el segundo parámetro el número de elementos que hay en la lista. Supongamos que el ejemplo de entrenamiento no tiene ninguna variable instanciada. Al aplicar el algoritmo EBG1 nos encontraremos con infinitas soluciones. En efecto, esto se produce ya que en el ejemplo de entrenamiento, la primera variable (la que representa a la lista) no está instanciada y por lo tanto puede ser unificada tanto con la lista vacía (`[]`) como con la lista `[Cabeza|Cola]`. Luego,

6. `[]` es la forma en que se denota una lista vacia.

existen dos definiciones del predicado `largolista` que pueden ser utilizadas para expandir el árbol de prueba. Al tener dos posibles expansiones para el árbol, tenemos dos posibles reformulaciones para el concepto objetivo, pero como una de las definiciones es recursiva, al expandirla, nuevamente tenemos dos posibles ramas y así sucesivamente. Gráficamente:

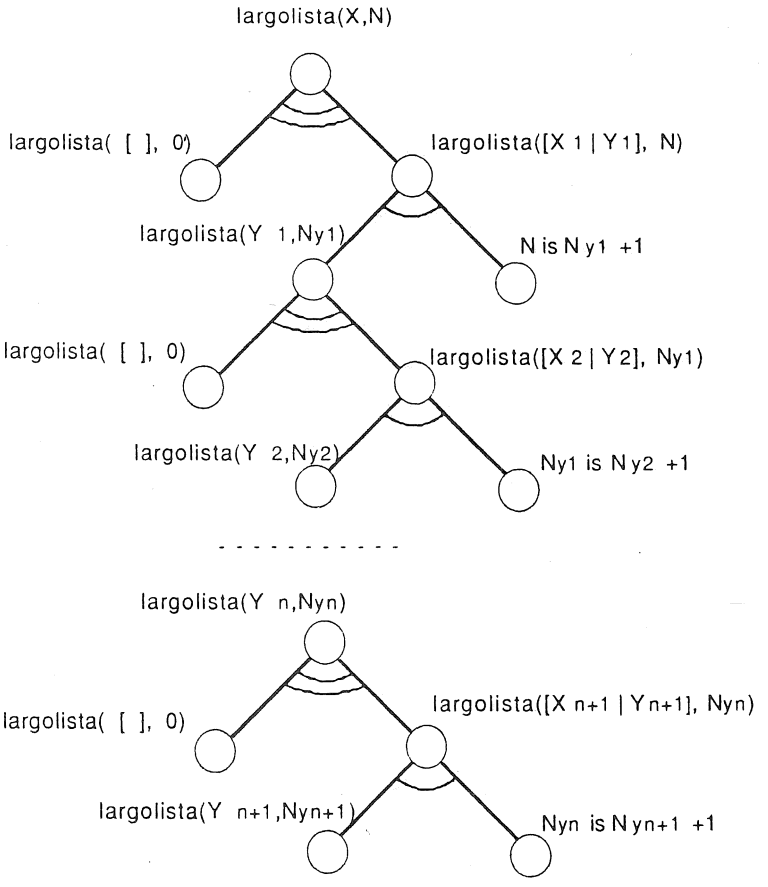


Figura 4.5: Infinitas soluciones en la expansión de `largolista`.

Esto hace imprecindible determinar cuales predicados son peligrosos de expandir al momento de construir el árbol de prueba para el ejemplo de entrenamiento. Una forma sencilla de hacer esto consiste (nuevamente) en dejar al usuario que especifique cuales predicados son peligrosos de expandir. Sin em-

bargo, aún nos queda planteada la duda de como hacerlo. Si lo hacemos en forma similar a la especificación de predicados operacionales, es decir, por medio de una lista, estaríamos siendo demasiado restrictivos ya que no todos los predicados `largolista(X,N)` son peligrosos de expandir. Por este motivo, lo más recomendable es especificar el predicado y la condición que lo hace peligroso. Por ejemplo, para `largolista` lo hacemos de la siguiente forma:

```
peligroso( largolista(X,N) ) :- \+ ground(X).
```
[7]

La incorporación de las extensiones mencionadas al algoritmo EBG1 nos lleva a la siguiente implementación para el algoritmo EBG2.

```
ebg2(Goal,GenGoal,[]) :- ejecutable(Goal),
                         call(Goal),
                         call(GenGoal),!.
ebg2(Goal,GenGoal,[GenGoal]) :- operacional(Goal),
                         peligroso(Goal),!.
ebg2(Goal,GenGoal,[GenGoal]) :- operacional(Goal),
                         call(Goal),!.
ebg2( (SubGoal,ListGoals), (SubgenGoal,ListgenGoal), Refor) :-
                         ebg2(SubGoal,SubgenGoal,Ref1),
                         ebg2(ListGoals,ListgenGoal,Ref2),
                         append(Ref1,Ref2,Refor).
ebg2( Goal, GenGoal, Ref) :-
                         clause(Goal,Body),
                         copy((Goal:-Body), (GenGoal:-GenBody)),
                         ebg2(Body,GenBody,Ref).
```
[8]

Figura 4.6: Implementación de EBG2.

7. `\+ ground(X)` es verdadero si X es una variable no instanciada

8. `append` recibe tres listas como argumentos, digamos `X`,`Y` y `Z`. El predicado es verdadero si el valor de `Z` es igual (sintacticamente) al valor de la concatenación de `X` e `Y`.

Nótese que se reemplazó la utilización del constructor de PROLOG (_ , _) del tercer argumento del predicado `ebg1` por la utilización de listas y el correspondiente predicado `append`.

5. Ejemplo

Hasta el momento, sólo se han presentado ejemplos bastante sencillos, en los cuales la reformulación del concepto objetivo no involucra mayores dificultades. A continuación, aplicaremos el nuevo algoritmo de EBG a un ejemplo más complejo, el que consiste en un intérprete de un lenguaje llamado Norma3, el cual recibe como entrada un programa que a su vez recibe datos de entrada. En este caso tenemos programas que trabajan en niveles de abstracción distintos. Gráficamente, esto lo vemos así:

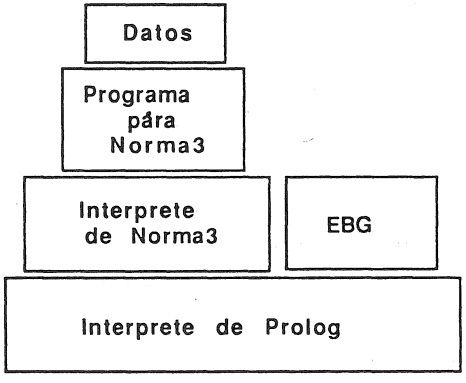


Figura 5.1: Niveles de abstracción en el ejemplo.

En este ejemplo, la teoría de dominio corresponde al conjunto de reglas y hechos que definen un interprete de Norma3, el que corresponde a un lenguaje imperativo que cuenta con cinco registros (identificados con las letras **v** a **z**), dos de ellos de entrada (**w** y **x**), dos internos o acumuladores (**v** e **y**) y uno de salida (**z**). En Norma3 sólo se manejan números enteros representados como una lista de símbolos "1" . Así, por ejemplo, el valor 3 será representado como [1, 1, 1]. El lenguaje permite ejecutar sentencias de asignación, operaciones matemáticas de suma, resta, saltos condicionales (if_then) y saltos incondicionales (goto´s). Además, Norma3 posee la capacidad de manejar listas de números. La cabeza, o

primer elemento de una lista, almacenada en un registro i se identifica con I, es decir, con la letra mayúscula que identifica el registro.

Este intérprete no cuenta ni con analizador léxico ni sintáctico, por lo que los programas escritos en Norma3 deben, por el momento, mantener el formato de listas de PROLOG. El nombre Norma3 obedece a motivos históricos, pues este interprete es una derivación de Norma2, lenguaje de similares características pero más simple [Fuller 89].

El concepto objetivo corresponde a la interpretación de un programa escrito en Norma3 que ordena una lista de mayor a menor usando un algoritmo de inserción. El código correspondiente a este programa es el siguiente:

```
0    [ y:=W,                          y = Cabeza de lista del registro w
1    x:=w,                            x = w
2    [ifX<y,1,1,1,1]                  IF Cabeza lista en x < y goto 4
3    y:=X,                            y = Cabeza de lista de registro x
4    x--,                             x = x - primer elemento
5    [ifx=0,1,1,1,1,1,1,1,1],         IF x = 0 goto 7
6    [goto,1,1],                      goto 2
7    w-y,                             w = w-y
8    v:=+y,                           v = v concatenado con la lista
                                      que contiene y
9    [ifw=0,1,1,1,1,1,1,1,1,1,1,1,1], IF w = 0 goto 11
10   [goto],                          goto 0
11   y:=v   ]                         y = v.
```

Figura 5.2: Programa sort en NORMA3.

Recordemos que la teoría de dominio utilizada por EBG corresponde a un intérprete, por lo tanto, los predicados operacionales, peligrosos y ejecutables pertenecerán al conjunto de reglas y hechos que definen a este intérprete. A continuación se presenta la declaración de estos predicados para el ejemplo:

```
operacional(Goal) :-
                    member(Goal,[without(_,_,_),length(_,_),
                                 add_one(_,_),sub_one(_,_)]).
operacional(Goal) :-
                    builtin(Goal).
```

[9]

9. `builtin(Goal)` es verdadero si `Goal` es un predicado predefinido del lenguaje.

```
ejecutable(Goal) :- clause(Goal,true),
                    \+ operacional(Goal).[10]
ejecutable(Goal) :- member(Goal, [skip(_,_,_), copy_val(_,_,_)).

peligroso( length(X,_) ) :-
                    \+ ground(X).
peligroso( Goal ) :-
                    builtin(Goal),
                    Goal =.. [_|Vars],
                    allground(Vars).[11]
```

Figura 5.3: Definición de meta-predicados para el ejemplo.

Así, se ve que los predicados `without`, `length`, `add_one` y `sub_one`, definidos en el intérprete de NORMA3, así como los predicados predefinidos de PROLOG, se considerarán como operacionales. Por otro lado, los predicados `skip` y `copy` de NORMA3 fueron definidos como ejecutables. Aquí, además se definió como ejecutable, cualquier predicado que cumpla con ser un hecho y no ser operacional.

Utilizaremos dos ejemplos de entrenamientos. En el primero, daremos como entrada al algoritmo EBG2, el programa de ordenamiento y una lista a ordenar, totalmente instanciada. Esto lo hacemos mediante el predicado `ebg2` de la siguiente forma, donde la variable `Pgm` ha sido instanciada previamente al programa sort en NORMA3.

```
ebg2(execute(Pgm, [ [1,1,1], [1,1,1,1], [1]], _, Salida),
        execute(Pgm, ListaIN, _, ListaOut), Reformulacion)
```

10. Todo predicado de la base de datos de PROLOG que sea un hecho y que no sea operacional es considerado ejecutable.

11 Esta última definición del predicado `peligroso` nos dice que no se debe expandir un predicado predefinido por el lenguaje cuando todos los argumentos no están instanciados. Esto es para evitar la expansión de cosas como `X > Y` ó `X is Y+1`.

Con esto, el resultado que se obtiene (una vez renombradas las variables y eliminadas las variables inútiles de la definición de `execute`, tal como `Pgm` que corresponde a una constante), es el siguiente:

```
execute_1( [Cabeza | Resto], [Mayor, Medio, Menor] ) :-
                length(Cabeza,N1),length(Cabeza,N2), N1 >= N2,
                sub_one([Cabeza | Resto],[Mayor | L1]),
                length(Mayor,N3),length(Cabeza,N4), N3 >= N4,
                sub_one([Mayor | L1],[Cabeza2 | Resto2]),
                length(Cabeza2,N5),length(Mayor,N6), N5 < N6,
                sub_one([Cabeza2 | Resto2],[]),
                without(Mayor,[Resto | Resto],[Medio | L2]),
                length(Medio,N7),length(Medio,N8), N7 >= N8,
                sub_one([Medio | L2],[Cabeza3 | Resto3]),
                length(Cabeza3,N9),length(Medio,N10), N9 < N10,
                sub_one([Cabeza3 | Resto3],[]),
                without(Medio,[Medio | L2],[Menor | L3]),
                length(Menor,N11),length(Menor,N12), N11 >= N12,
                sub_one([Menor | L3],[]),
                without(Menor,[Menor | L3],[]).
```

Figura 5.4: Resultado del ejemplo.

Nótese que esta redefinición del predicado `execute` en términos de los predicados operacionales, cumple con ordenar una lista de tres elementos siempre y cuando estén en el mismo orden de la lista del ejemplo de entrenamiento. En otras palabras, se aprendió como ordenar una lista con ciertas características. En este caso, el tipo de listas que se aprendió a ordenar es bastante específico.

Intentemos un segundo ejemplo de entrenamiento, que nos permita ordenar cualquier lista de tres elementos. Para ello, utilizaremos el mismo programa de ordenamiento, pero esta vez la lista a ser ordenada estará instanciada sólo en la estructura, es decir, aplicaremos

```
ebg2(execute(Pgm, [ X, Y, Z], _, Salida),
        execute(Pgm, ListaIN, _, ListaOut), Reformulacion)
```

donde `X`, `Y` y `Z` son variables que están sin instanciar. El resultado que se obtiene es un conjunto de 64 definiciones para el predicado `execute`, el que es capaz de ordenar cualquier lista de tres elementos. Es importante destacar que la reformulación del ejemplo anterior contiene seis comparaciones y las 64 redefiniciones corresponden a todas las posibles combinaciones que se producen al realizar seis comparaciones (pues 2 elevado a 6 es 64).

6. Conclusiones

Se ha presentado una extensión al algoritmo original de aprendizaje basado en ejemplos, obteniendo generalizaciones con menor número de clausulas y por lo tanto más eficientes. Las extensiones al algoritmo tradicional, incorporándoles meta-predicados para dirigir el proceso de construcción del árbol de prueba permitieron la experimentación con ejemplos de gran tamaño, lo que no es posible con los algoritmos actuales.

Sin embargo, es necesario profundizar el estudio de los criterios para determinar cuales predicados pueden ser considerados operacionales, ejecutables o peligrosos. En este artículo, dichos predicados deben ser definidos por el usuario, lo que no siempre es factible. En este último aspecto, es posible beneficiarse de las similitudes entre EBG y evaluación parcial [van Harmelen y Bundy 88] [Fuller y Hoppe 91] para automatizar la detección de predicados peligrosos [Fuller 91]. Específicamente, se estima que la definición de los predicados `peligroso` y `ejecutable` podrían ser hechas en forma automática por el mismo sistema de aprendizaje, basado en técnicas de interpretación abstracta [Abramsky y Hankin 84].

Con esta técnica de aprendizaje es posible obtener mejoras en la eficiencia de la ejecución de los programas producidos de un orden de magnitud, con lo que la utilidad de la técnica queda de manifiesto. Sin embargo, existe una fuerte limitación asociada a la representación del conocimento. En efecto, al utilizar una representación del conocimiento basada en lógica de primer órden, sólo se pueden hacer generalizaciones en este contexto y quedean excluidas representaciones asociadas al razonamiento no-monótono o al tratamiento del tiempo.

Otra aplicación de esta técnica en la que se está trabajando es en el campo del lenguaje natural. Ahí, se piensa aplicar la técnica de EBG para explorar la idea de "tutores inteligentes" del lenguaje. En este caso, el algoritmo de EBG sobre una interfaz de lenguaje natural y una sentencia para dicho lenguaje podría construir el árbol de prueba. En caso de que la prueba no logre llegar a las hojas del árbol, significa que la sentencia entregada por el usuario fue incorrecta, y el sistema podría corregirla partiendo de los nodos no expandidos del árbol.

Agradecimientos

Este trabajo ha sido financiado gracias al apoyo del Fondo Nacional de Ciencia y Tecnología de Chile, FONDECYT, proyectos 90-0688 y 89-0591 y al proyecto CHI/88/022 del Programa de las Naciones Unidas para el Desarrollo.

Bibliografía

1. Abramsky, S. y Hankin, C. (eds.), Abstract Interpretation of Declarative Languages, Wiley (1987).
2. Bratko, I., PROLOG Programming for Artificial Intelligence, segunda edición, Addison-Wesley (1990).
3. Fuller, D., Partial Evaluation and Mix Computation in Logic Programming, tésis de PhD, Department of Computing, Imperial College of Science and Technology, Londres, Inglaterra (1989).
4. Fuller, D., Replacing the Loop Detection Scheme in Partial Evaluation of Logic Programs, en preparación (1991).
5. Fuller, D. y Hoppe, U., Explanation Based Generalization and Partial Evaluation Revisited, enviado a Artificial Intelligence (1991).
6. van Harmelen, F. y Bundy A., Explanation-Based Generalisation = Partial Evaluation, Artificial Intelligence 36 (1988).
7. Kedar-Cabelli, S.T. y McCarthy, L.T., Explanation-based Generalization as a resolution theorem proving, Proceedings of the Fourth International Machine Learning Workshop, Irvine, EE.UU. (1987).
8. Lloyd, J., Foundations of Logic Programming, Springer-Verlag (1984).
9. Mitchell, T.M., Keller, R.M. y Kedar-Cabelli, S.T., Explanation-based Generalization: A unifying view, Machine Learning Vol I (1986).

10. Michalski R.S., Understanding the nature of learning, Machine Learning Vol II (1986).
11. Simon, H.A., Why Should Machines Learn?, en Machine Learning: An Artificial Intelligence Approach, R.S. Michalski, J.G. Carbonell y T.M. Mitchell (Eds). (1983).

DISEÑO Y EVALUACION DE UN SERVIDOR DE RECURSOS PARA UNA RED LOCAL*

Francisco Aurtenechea O.
Ignacio Casas R.
Roberto Cumsille U.
Marco Kútulas. P

Pontificia Universidad Católica de Chile
Depto. de Ciencia de la Computación
Casilla 6177, Santiago, Chile
FAX: (562) 5524054 email: fa@puc.cl

RESUMEN

En este trabajo se presenta el diseño y análisis de un sistema de acceso transparente a servicios distribuídos en una red local Ethernet de la Universidad Católica de Chile. El sistema está soportado por un conjunto de primitivas de comunicación de alto nivel, cuyo objetivo es facilitar el desarrollo de sistemas distribuídos, especialmente en relación al acceso transparente a recursos de una red local. El sistema consta actualmente de una interfaz de comunicación entre procesos remotos, tanto para mini-computadores Unix como micro-computadores PC-DOS, y una interfaz de usuario para micro-computadores PC-DOS. Ambas interfaces contienen primitivas que pueden ser invocadas desde el lenguaje C. Se incluye un análisis del desempeño del sistema en el acceso a aplicaciones remotas, y se compara con el acceso local.

I. INTRODUCCION

En ambientes universitarios es muy común encontrar una gran variedad de equipamiento computacional. La tendencia es hacia la provisión de herramientas que faciliten la interconexión de estos equipos mediante el uso de redes locales, a diferentes niveles de servicio. Entre estas herramientas se incluyen tanto hardware estándar de conexión (p.ej.: tarjetas Ethernet) como mecanismos simples de intercambio de información vía software (p.ej.: login remoto, correo electrónico, transferencia de archivos).

Sin embargo, la programación de aplicaciónes más sofisticadas, especialmente aquellas que involucran procesamiento distribuído [Che88, Tan85] requieren de un alto grado de conocimiento de los sistemas operativos en los cuales se requiere correr la aplicación. Además, el programador debe involucrarse con estructuras de datos y primitivas de comunicacíon de bajo nivel que son distractoras del objetivo que se persigue al construir una aplicación. Normalmente, para permitir la conexión estre dos

* Este trabajo ha sido financiado por FONDECYT, proyecto 0187/91

o más procesos remotos o el envío de mensajes entre ellos, es necesario escribir un trozo de código no trivial compuesto por llamados a rutinas del sistema operativo, p. ej.: mediante el uso de una biblioteca de sockets [Exc88].

Por otro lado, en la medida que las redes se expanden, evolucionan y la demanda por su uso aumenta, es importante conocer las limitaciones en el desempeño que imponen a las aplicaciones de red. Por ende, los datos de estudios de desempeño son una evidencia importante para estrategias de diseño y planificación de capacidad.

El propósito de este trabajo es proporcionar un medio ambiente de software para el programador de aplicaciones en el ámbito de sistemas distribuídos, donde coexisten sistemas heterogéneos, incluyendo sistemas multiusuarios y micro-computadores conectados en una red local Ethernet [Met76]. El sistema está orientado hacia la provisión de una interfaz de comunicación homogénea, de alto nivel, compuesta por un conjunto de primitivas de comunicación entre procesos que permitan hacer transparente el procedimiento de comunicación particular de cada sistema. El análisis de desempeño del sistema, mediante una herramienta prototipo, se presenta también como un objetivo de este trabajo, y se considera a futuro la provisión de herramientas de desempeño de carácter general para redes locales.

Como base de desarrollo, se diseñó un núcleo de comunicación compuesto por un conjunto de primitivas de enlace entre procesos remotos. Estas permiten esconder al programador los problemas inherentes al establecimiento de sesiones entre procesos de la red, presentando un nivel de abstracción mayor que las herramientas tradicionales de comunicación entre procesos. De este modo, se pretende simplificar el desarrollo de sistemas de acceso a recursos distribuídos, liberando al programador de los detalles de bajo nivel en la comunicación entre procesos de una red. El objetivo es incrementar la productividad del programador y potenciar la gama de posibilidades para el uso de los recursos que ofrece la red, especialmente a microcomputadores conectados a ésta.

El desarrollo de estas rutinas se sustenta en el uso de las herramientas proporcionadas por la biblioteca estándar Socket Library [Exc88], y que están disponibles actualmente en la mayoría de los computadores de la red. A nivel de comunicación en la red, se utilizan protocolos de comunicación estándar tal como TCP/IP. Sin embargo, estas facilidades son transparentes para el programador ya que

operan en los niveles de transporte y de red dentro del modelo de red OSI [Iso81], definido para arquitecturas de comunicación abiertas.

Mediante el núcleo de comunicación desarrollado se contruyó un sistema de acceso transparente a los recursos disponibles en distintos equipos Unix de la red, bajo el esquema **cliente/servidor** [ver Tan85], incorporando como clientes tanto a equipos Unix como equipos MS-DOS. En beneficio de la simplicidad en la provisión de los numerosos potenciales servicios de la red, en cada nodo se incorpora un único **servidor de procesos** que activa a un proceso específico cuando un cliente solicita un servicio.

El sistema fue concebido, fundamentalmente, para ser usado en un ambiente universitario, donde los requerimientos computacionales, además de que normalmente no involucran grandes volúmenes de datos, son del más variado tipo: compilación, acceso a sistemas de bases de datos, uso de sofware estadístico, edición de documentos, impresión de archivos, etc.

Los clientes Unix proveen una interfaz de usuario simple, en base a un único comando. En los micro-computadores la interfaz de usuario está compuesta por un conjunto de rutinas gráficas que permiten desarrollar un esquema de "pull down" menús y ventanas. Este sistema permite tener varias aplicaciones remotas concurrentes en la pantalla del PC, cada una desplegando su salida en una ventana independiente y en forma transparente a la red.

Finalmente, aprovechando las herramientas construídas, se diseñó un prototipo de sistema de evaluación del desempeño en el acceso a servicios remotos basados en el modelo cliente/servidor. Se incluyen resultados comparativos de tiempos de acceso remoto y local para algunas aplicaciones.

II. NUCLEO DE COMUNICACION

En los útimos años se han diseñado algunos modelos de comunicación para proveer interconectvidad a nivel de procesos en una red [ver Tan85]. Uno de los modelos más utilizados es el de paso de mensajes. Se basa en la existencia de procesos que administran recursos remotos (servidores) y procesos que solicitan estos recursos

(clientes). El desarrollo de una aplicación distribuída dentro del modelo anterior requiere de la incorporación de dos módulos: un módulo servidor en la máquina donde se encuentra el recurso, y un módulo cliente en la máquina desde donde se quiere hacer uso del recurso remoto.

Basado en el modelo anterior, se desarrolló un conjunto de primitivas de comunicación, conformando un núcleo de comunicación cuyo objetivo es proporcionar una plataforma base, independiente del sistema operativo del computador, para facilitar el desarrollo de aplicaciones en ambiente de red local.

El núcleo presupone la provisión de software estándar de comunicación tanto a bajo nivel en la red (por ejemplo, protocolos basados en IEEE 802.3) como en sus niveles intermedios (por ejemplo, protocolo TCP/IP). Esta suposición no disminuye la flexibilidad del sistema, por cuanto la disponibilidad de estas herramientas está ampliamente difundida en la mayoría de los equipos que coexisten en una red local en la actualidad (UNIX, VMS, MS-DOS, Macintosh, etc.).

TCP e IP son protocolos complejos que requieren de una cantidad considerable de procesamiento, y le imponen una alta carga al computador en términos de consumo de memoria, tiempo de procesamiento, y número de interrupciones. Por este motivo, para utilizar estos protocolos, especialmente desde un PC, sin recargar su CPU es conveniente usar una tarjeta de comunicación del tipo "inteligente". Esta tarjeta debe tener suficiente memoria para procesar el protocolo TCP/IP y manejar buffers para la comunicación con la red. De esta forma se alivia a la CPU de la alta carga que representa el procesamiento de comunicación, aumentando así la eficiencia global del sistema.

Por otro lado, la interfaz de comunicación se sustenta en el uso de **sockets** [Exc88] como conjunto de rutinas de comunicación entre procesos. Esta herramienta, inicialmente incorporada en sistemas UNIX, se provee actualmente en la mayoría de los computadores, incluyendo recientemente micro-computadores. El núcleo de comunicación desarrollado, sin embargo, presenta al programador un nivel mayor de abstracción que el proporcionado por estas rutinas de comunicación, facilitando el proceso de desarrollo de aplicaciones en sistemas distribuídos.

2.1 Interfaz de Comunicación

La interfaz de comunicación ha sido diseñada como un conjunto de primitivas de alto nivel. Estas primitivas están agrupadas en dos bibliotecas de rutinas en lenguaje C. Una de ellas incorpora las primitivas para desarrollar un programa cliente tanto en ambiente Unix como DOS. La otra incorpora las primitivas para el desarrollo de un programa servidor en máquinas multiusuario (p.ej.: Unix).

Las primitivas de comunicación para un **programa cliente** son las siguientes:

connect_to_server (char *service, *hostname)

Inicia una conexión con un servidor. Recibe como parámetros el nombre del servicio (service), y el nombre del host donde éste reside (hostname). Como resultado entrega un identificador numérico, de tipo integer, del socket (**sock**) sobre el cual se realiza la conexión y que corresponde a un subproceso de atención creado por el servidor.

disconnect (int *sock)

Esta función permite cerrar una conexión establecida con un servidor. Recibe como parámetro el identificador del socket (sock). Esta función no devuelve un resultado.

read_from_server (int sock, char *buffer, int buffersize)

Esta función es utilizada para leer información enviada por el servidor a través de la red. Recibe como parámetros el identificador del socket (sock), el buffer donde se almacenarán los datos leídos (buffer), y el tamaño del buffer (buffersize). Si la operación se ejecutó exitosamente, entrega como resultado la cantidad de caracteres recibidos, en caso contrario devuelve -1 y despliega un mensaje de error en la pantalla.

write_to_server(int sock, char *buffer, int n)

Esta función permite enviar datos al servidor. Recibe como parámetros el identificador del socket (sock), el buffer de datos a enviar (buffer), y el número de caracteres a enviar (n). En caso de éxito en el envío de los datos, esta función entrega el número de caracteres transmitidos por la red, en caso contrario devuelve -1 y despliega un mensaje de error en pantalla.

block_socket (int sock)

Esta función deja el socket en modo de lectura síncrona. Esto significa que la función read_from_server retorna el control al programa sólo cuando se reciben datos desde la red. Recibe como parámetro el identificador del socket (sock).

unblock_socket (int sock)

Esta función es la inversa de la anterior, es decir, deja el socket en modo de lectura asíncrona. Esto significa que la función read_from_server retorna el control al programa, el cual si no tiene datos que leer en ese instante puede continuar con otra operación y/o re-intentar una lectura posteriormente

Set_Confirm (int sock, flag).

Esta función sirve para establecer confirmaciones de mensajes. Recibe como parámetros, el identificador del socket (sock), y una valor entero (flag) que indica ya sea confirmar (flag=1) o no (flag=0) los mensajes desde y hacia el servidor. (por omisión flag=0).

Un programa tipo cliente debe, en primer lugar, conectarse con un servidor, indicándole el servicio requerido. Una vez establecida la conexión se procede al intercambio de información a través de la red, hasta que el cliente cierre la conexión. En la fig. 1 se muestra un esquema básico de un programa cliente.

```
#include "clien_lib.h"                          /* interfaz de comunicación */
main()
{
.....
sock = connect_to_server(service, hostname);  /* conexión con el servidor */
.....
while (...)                                     /* loop con servidor remoto */
      {
          ......
          read_from_server(sock,buffer,buffersize);
          ......
          write_to_server(sock,buffer,n);
          ......
      }
      disconnect(sock);
}
```

Figura 1. Esquema de construcción de un programa cliente

La biblioteca de primitivas disponibles para un programa servidor permite desarrollar un programa que corriendo en una máquina multi-tarea puede atender concurrentemente a varios clientes ubicados en distintos nodos de la red.

Las primitivas de comunicación para un **programa servidor** son las siguientes:

run_server (char *service, *<nomproc>)

Esta función inicializa un servidor y lo deja esperando por requerimientos desde los clientes. Al recibir una conexión se genera un subproceso que atiende al cliente respectivo, volviendo el servidor a esperar por un requerimiento de otro cliente. Recibe como parámetros el nombre del servicio a ofrecer (service), y el nombre de la rutina que lo provee (<nomproc>) la cual es invocada desde cada subproceso. Esta última es desarrollada por el programador y puede hacer uso de las primitivas del núcleo. La función run_server retorna un valor entero indicando el éxito o fracaso de la operación.

read_from_client (char *buffer, int buffersize)

Esta función es utilizada para recibir información enviada por el cliente a través de la red. Recibe como parámetros el buffer donde se almacenan los datos recibidos (buffer), y el tamaño del buffer (buffersize). Como cada subproceso atiende sólo a un cliente, no es necesario indicar la identificación del socket correspondiente. Si la operación se ejecutó exitosamente, entrega como resultado la cantidad de caracteres recibidos, en caso contrario devuelve -1 y despliega un mensaje de error.

write_to_client (char *buffer, int n)

Esta función envía datos al cliente. Recibe como parámetros el buffer con los datos a enviar (buffer), y el número de carácteres a transmitir (n). Si la operación se ejecutó exitosamente, entrega como resultado el número de carácteres enviados, en caso contrario devuelve -1 y despliega un mensaje de error.

block_socket

Esta función deja el socket de conexión con el cliente en modo de lectura síncrona y es similar a la rutina incluída para un programa cliente.

unblock_socket

Esta función deja el socket de conexión con el cliente, en modo de lectura asíncrona y es similar a la rutina incluída para un programa cliente.

<nomproc>

Esta función debe ser desarrollada por el programador, y contiene el código de la aplicación, la cual otorga un determinado servicio a los clientes de la red (el sistema servidor de procesos, que se describe en la sección 3, fue implementado bajo este esquema). Es una rutina que corre como un subproceso del servidor, y es activada una vez que un cliente ha solicitado una conexión con dicho servidor.

Un programa servidor consta de dos partes. La primera inicializa el proceso servidor el cual queda a la espera de los requerimientos de clientes, y la segunda corresponde a la función <nomproc> mencionada arriba, la que contiene el código de atención al cliente haciendo uso de las primitivas de comunicación. En la fig. 2 se muestra un modelo de un programa servidor.

```
#include  "serv_lib.h"                    /* interfaz de comunicación */
main()
{
      run_server(service,  ServProc);     /* inicializa el servidor   */
}

ServPro:                                  /* desarrollo del servicio */
{
      while (...)                         /* loop de atención al cliente  */
      {
            .......
            read_from_client(buffer,buffersize);
            ......
            write_to_client(buffer,n);
            .......
      }
      exit();                             /* finaliza el subproceso */
}
```

Figura 2 Esquema de desarrollo de un servicio

III.- SERVIDOR DE PROCESOS

Utilizando las rutinas de propósito general descritas en la sección anterior, se desarrolló un sistema de acceso a recursos remotos, disponibles en los distintos computadores de la red (compiladores, bases de datos, comandos del sistema operativo, etc).

Cada nodo del sistema incluye dos módulos básicos, el módulo **cliente** y el módulo **servidor de procesos**. El primero es un proceso que permite invocar cualquier aplicación remota. El segundo es un proceso que, a través de una puerta de comunicación fija, escucha por requerimientos de clientes que solicitan servicios del nodo. Esto evita tener un servidor activo en la red por cada tipo de recurso que pueda ser invocado remotamente (ver fig. 3).

El servidor de procesos, una vez que recibe un requerimiento de conexión, crea un subproceso para luego quedar en espera de otro cliente. Este nuevo subproceso recibe el nombre del servicio requerido, y lo ejecuta, comunicando la entrada (input) y la salida (output) de dicho servicio al proceso cliente en el nodo remoto.

El esquema de administración de recursos es distribuído. En cada nodo existen dos archivos que permiten a un cliente obtener información de los nodos y sus aplicaciones disponibles. El primer archivo contiene una lista de las aplicaciones que pueden ser invocadas en el sistema junto con el nodo donde residen. El segundo, contiene los nombres de los distintos nodos de la red y la puerta de acceso al servidor de cada nodo. De esta forma es posible expandir o restringir el sistema a nodos y aplicaciones específicos para distintas máquinas.

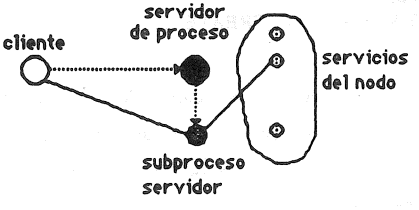


Figura 3: Esquema general del sistema.

Por sobre los protocolos básicos de la red (por ej.: IEEE802.3, TCP/IP) se incluye, opcionalmente, un protocolo del tipo **stop-wait**, útil en sistemas que requieran mayor confiabilidad (ver primitiva SetConfirm arriba). Este protocolo con una pérdida de eficiencia muy pequeña, permite asegurar la confiabilidad en los datos en un ambiente donde coexisten sistemas heterogéneos y no necesariamente confiables.

3.1 Utilización de Procesos

El módulo cliente para equipos Unix incluye dos procesos que manejan separadamente el *input* y *output* del terminal del usuario. Con esto se obtiene un mejor desempeño y una adecuada sincronización para ambas actividades. Los sistemas MS-DOS, al ser del tipo mono-tarea, requieren de adaptaciones en relación al número de procesos. En este caso un sólo proceso maneja el flujo de datos en ambos sentidos.

En el servidor de procesos, existe un solo proceso que interactúa con el cliente. De esta manera se evita una proliferación excesiva de procesos en el servidor ya que éste puede ser invocado concurrentemente por muchos clientes.

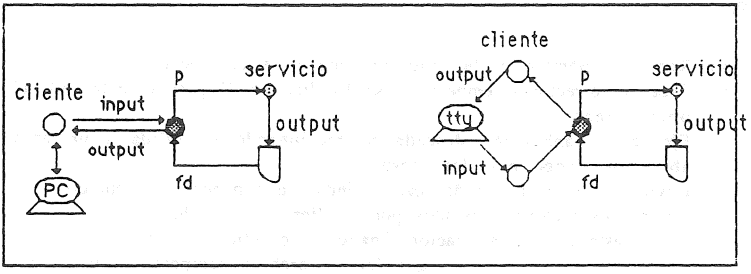
El mecanismo básico utilizado por el servidor de procesos, para otorgar acceso a aplicaciones remotas, es mediante la función **popen** que provee UNIX. Esta función permite abrir un "pipe" a un proceso para leer o escribir sobre él, realizar I/O redireccionando la salida de la aplicación, para luego escribir los comandos sobre ella en el identificador de archivo que retorna la función popen.

Luego, con las siguientes instrucciones se puede tener acceso a la aplicación en forma sencilla (en este ejemplo el servidor de aplicaciones invoca al intérprete FLISP):

```
p = popen ( "Flisp > output" , "w" );
fd = open ("output" , O_RDONLY | O_CREATE );
```

Luego, la aplicación remota (Flisp) recibe los datos del cliente mediante el identificador **p** del "pipe" y escribe sus resultados en un archivo temporal (output) donde el servidor (en este caso un subproceso del servidor de procesos) puede leerlos usando el identificador de archivo **fd**. Este procedimiento se puede apreciar en la fig. 4, tanto para equipos Unix como equipos MS-DOS.

Una vez que se obtiene acceso de I/O con la aplicación, es sencillo conectar este I/O con el cliente. Para esto se tiene una iteración indefinida en que todo el *output* de la aplicación se transmite al cliente por la red y todos los comandos que el cliente envía se escriben en el *input* de la aplicación.



a) sistema MS-DOS b) sistema Unix

Figura 4 Esquema del acceso a una aplicación remota

Esta solución es satisfactoria, ya que permite ejecutar aplicaciones remotas redireccionando el I/O de un proceso hacia el cliente. Esto se validó corriendo algunas aplicaciones como LISP, ORACLE, SCHEME, TELNET, y comandos Unix tales como who, ps, ls, etc. El sistema desarrollado se compone actualmente de servidores de procesos que corren en máquinas Unix y clientes que corren en equipos Unix y micro-computadores MS-DOS.

Los pasos que se verifican en el **proceso cliente**, cada vez que el usuario selecciona un servicio son los siguientes :

a) Conexión con el servidor.
b) Si el servidor responde, se le envía el nombre de la aplicación a ejecutar.
c) El cliente espera una señal del servidor que indica que éste posee recursos suficientes para atenderlo. De no ser así, se aborta la comunicación.
d) Se envían los archivos necesarios para ejecutar la aplicación remota (p.ej. un programa a compilar). Luego, el cliente itera hasta que ocurra uno de los eventos siguientes:
   - El servidor indica el término de la aplicación remota.
   - El cliente envía un comando de término de conexión
   - El usuario activa el menú principal (caso de MS_DOS).
   - El usuario activa otra ventana (caso de MS-DOS).
e) Finalmente se reciben los archivos modificados o generados durante la ejecución de la aplicación.

Con respecto al **servidor de procesos** de un nodo, éste está permanentemente escuchando en una puerta si existe un cliente que desea conectarse. Al recibir este requerimiento, el servidor procede a crear un subproceso que será el encargado de atender al cliente respectivo, mientras el servidor sigue esperando a otro cliente. Este nuevo proceso generado ejecuta la rutina **<nomproc>**, la que realiza las siguientes operaciones:

a) Recibe el nombre de la aplicación que se desea ejecutar.
b) Crea un directorio temporal y se localiza en él para ejecutar allí el resto de las operaciones.
c) Ejecuta la aplicación deseada redireccionando su salida a un archivo temporal, abriéndolo luego para leer desde él.
d) Envía al cliente una señal que le indica que puede ser atendido.
e) Recibe los archivos enviados por el cliente (si los hay).
f) Itera hasta que la aplicación finalice o el cliente le envíe una señal de término. En este proceso, el servidor lee el archivo temporal, output de la aplicación, enviándole al cliente toda la información que allí exista. Luego se leen los comandos que envía el cliente para comunicárselos a la aplicación.
g) Finalmente, al término del proceso se envían de vuelta los archivos modificados o creados durante la ejecución de la aplicación (si los hubo), y se procede a borrar el directorio temporal.

4 INTERFAZ DE USUARIO

La interfaz de usuario para acceder al sistema anterior fue desarrollada para dos ambientes, uno simple al estilo de comandos Unix y otro más sofisticado y "amistoso" orientado a micro-computadores.

4.1 Interfaz de comando

El sistema servidor de procesos está actualmente instalado en equipos Unix V y Unix 4.2 BSD, con diferencias mínimas en su implementación. Una interfaz simple de usuario disponible tanto en clientes de terminales ASCII como de PCs consiste de un comando que conserva la sintaxis típica de los comandos en Unix. En su expresión más simple, el comando es:

net <nombre-servicio>

Donde, <nombre-servicio> es el nombre del servicio remoto a ejecutar (p.ej: compilador, editor, comando). En su expresión más compleja, el comando es:

net [-c] [-n nodo] <nombre-servicio> [-f <lista de archivos>]

La opción "-n nodo" indica el lugar (nodo) de ejecución del servicio invocado. Esta opción es útil cuando un servicio reside en varios nodos. La opción -c se usa para aquellas aplicaciones, que poseen comandos de una sola tecla (por ejemplo editores como **emacs** o **vi**), que requieren que cada tecla presionada por el usuario sea comunicada inmediatamente a la aplicación.

La opción -f se usa para especificar el uso de archivos, indicando la separación entre la aplicación a usar y los archivos a transferir. Con esta opción se evita la complejidad de manejar la transferencia de archivos con posterioridad a la invocación de la aplicación, en aquellas aplicaciones que así lo hacen normalmente. Algunos ejemplos de este comando se muestran a continuación:

```
net -n juncal who              (consulta de usuarios activos en nodo juncal)
net -c emacs new.c             (edición de new.c en un editor emacs de la red)
net Lisp -f  x.lsp  y.lsp      (uso de intérprete Lisp de la red con dos archivos)
net sqlplus -f A.sql  B.sql    (uso de base datos sqlplus de la red, con dos archivos)
```

4.2 Interfaz para PCs

Una Interfaz de usuario basada en un sistema de ventanas permite hacer un uso más eficiente de la pantalla de un computador. Debido a la posibilidad de tener varias ventanas concurrentemente en la pantalla, cada una desplegando la salida de una aplicación remota distinta, es que este tipo de interfaz es muy adecuada en ambientes de red.

Mediante esta interfaz se puede tener acceso tanto a aplicaciones remotas como a dispositivos de gran capacidad, tales como discos, impresoras, servidores de comunicación, etc., transparente a su localización en la red y forma atractiva al usuario. Las aplicaciones que corren concurrentemente en las distintas máquinas de la red, despliegan sus resultados en una ventana independiente en la pantalla del PC.

Para construir la interfaz se desarrollaron diversas rutinas de manejo de ventanas, y menús, utilizando el lenguaje C y las rutinas gráficas de Quick-C. Mediante estas rutinas se provee actualmente un prototipo de interfaz de usuario que incorpora

diversas facilidades para el acceso a recursos remotos. Cada ventana abierta en la pantalla representa una conexión con un subproceso servidor de la red.

La interfaz provee un mecanismo para que el usuario pueda ir activando las ventanas abiertas. Esto le permite pasar de una aplicación remota a otra con sólo presionar una tecla. Con esta facilidad el usuario puede dejar, por ejemplo, corriendo una larga compilación en una ventana, imprimiendo un documento en otra, mientras hace consultas a una base de datos remota en la ventana activa. Mediante otra tecla, es posible dejar superpuestas las ventanas, de modo que la ventana activa sea la única que se muestra visible, y a un tamaño adecuado para la aplicación.

La fig. 5 presenta un esquema simplificado de esta interfaz, donde se muestra una sesión real de uso de recursos de la red: usando un intérprete de la red en la ventana activa, en otra realizando una conexión a un computador de la red, y en otra ventana invocando un comando remoto. El esquema de menús es jerárquico, en donde existe una subdivisión por categorías de recursos en la red (comandos, aplicaciones, lenguajes, etc). Por ejemplo, la categoría lenguajes de programación, se subdivide en lenguajes funcionales (p.ej.: FP, LISP, Scheme, etc), imperativos (p.ej.: Pascal), lógicos (p.ej.: Prolog), etc. En caso de existir una aplicación residente en más un nodo, la interfaz consulta al usuario por el nodo respectivo, y en caso de omisión el sistema escoge uno.

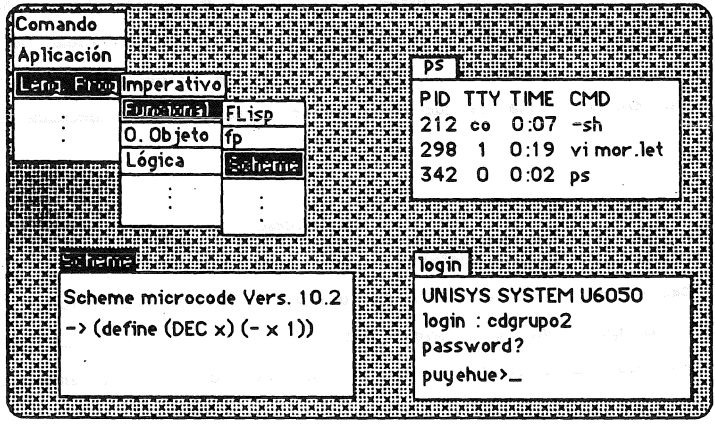


figura 5. Esquema de interfaz de usuario en PC

Actualmente, cada módulo cliente mantiene una pequeña base de datos con las aplicaciones de la red y los nodos donde residen, involucrando así una administración descentralizada de recursos. Se considera que ésta es adecuada para el tamaño actual de la red (no más de 10 nodos). En la medida que la red se expanda se pretende optar por un esquema administrativo que involucre la existencia de agrupaciones de nodos, por ejemplo de acuerdo a un patrón de actividad común de los nodos, asignándole a cada grupo un proceso localizador [Aur90], el cual queda residente en alguno de los nodos que conforman el grupo.

V ANALISIS DE DESEMPEÑO

El sistema desarrollado se enmarca dentro de un esquema de acceso del tipo cliente/servidor. Este es un modelo típico de interacción entre computadores conectados en red local, y es especialmente útil a estaciones de trabajo con pocos recursos, conectadas a la red. La tendencia, hoy en día, es hacia un aumento progresivo de equipamiento computacional conectado en redes locales, donde coexisten algunos equipos que actúan como servidores y especialmente equipos que son clientes de los primeros.

Por ende, es importante analizar el impacto que la red, a través de sus diversos componentes, representa en el desempeño de un sistema tipo cliente/servidor. Con esta motivación, se procedió a desarrollar un sistema simple de medición de desempeño, mediante la incorporación de rutinas, en el módulo cliente, que permitieran tanto la detección de tiempos reales de respuesta desde el servidor como la generación controlada de requerimientos desde el cliente.

En las mediciones se escogieron fundamentalmente aplicaciones remotas interactivas, de modo de medir el impacto de la red antre una consulta y la llegada de la respuesta correspondiente. Para ello, además de las mediciones de tiempo en el cliente, se procedió a medir paralelamente los tiempos de respuesta de la aplicación ante consultas locales, para tener una medida del comportamiento del sistema independiente de la aplicación siendo utilizada. Además, las mediciones se realizaron en períodos de baja carga (p.ej. durante la noche), y con varias repeticiones para aislar el impacto de carga externa al sistema.

Una de las aplicaciones escogidas fue el sistema administrador de base de datos SQLPLUS, localizado en un computador Unix con procesador Intel 80386, seleccionándose consultas típicas tales como Select, Join y Help, a una base de datos pequeña. Con el fin de analizar la degradación del tiempo de respuesta frente a un aumento de carga en el servidor, se procedió a generar, automaticamente, distintas cargas de trabajo (reales), desde un cliente, consistentes en usuarios invocando una determinada consulta a través la red.

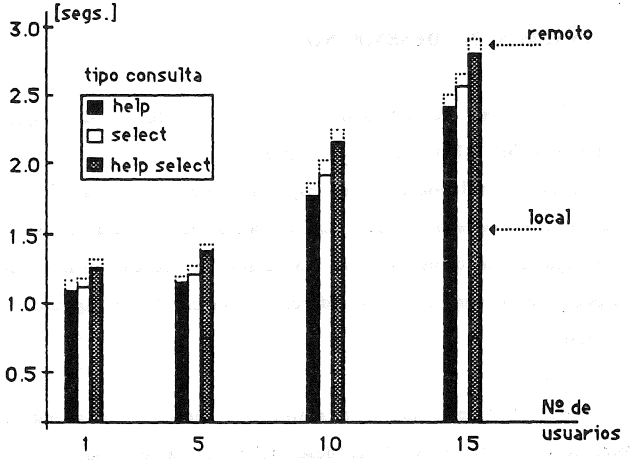


Fig. 6 Tiempos de respuesta en acceso a una base de datos

La fig. 6 muestra el tiempo de respuesta promedio a consultas tanto locales como remotas. Se puede apreciar que la degradación que produce el sistema de acceso remoto es relativamente pequeña. Por otro lado, aunque no presentado en la fig. 6, para una carga de más de 20 usuarios, se observó una degradación significativa en el tiempo de respuesta, pero ésta era debida exclusivamente a la saturación de la máquina que corría la aplicación.

Por otro lado, para el caso de acceso remoto se realizó un análisis de degradación porcentual en los tiempo de respuesta. El resultado de ese análisis se muestra en la fig. 7. Se aprecia que existe un mayor incremento en el tiempo de respuesta, al aumentar de aproximadamente 5 a 6 usuarios. Esto se podría explicar debido a que el software de la red habilita, actualmente, colas de atención simultánea

de hasta 5 clientes, debiendo los clientes en exceso de esta cantidad repetir la operación de acceso a la cola.

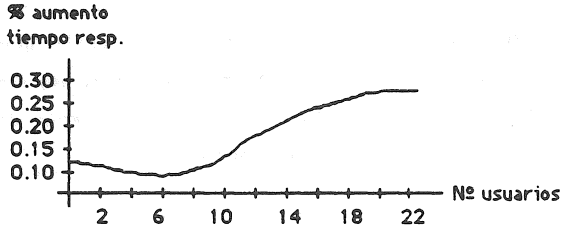


Figura 7. Variación porcentual en tiempo de respuesta remoto

Los resultados mostrados en esta sección, no pretenden mostrar un análisis exhaustivo de medición del desempeño del sistema. Su objetivo fue por un lado, entregar una visión general de su comportamiento frente a algunos requerimientos, y por otro construir un primer prototipo de evaluación de desempeño susceptible de mejorar a futuro.

Como próxima etapa, se proyecta incorporar herramientas de monitoreo en el núcleo de comunicación descrito en la sección 2, para adaptar el prototipo anterior a un sistema que permita desagregar el análisis del desempeño en distintos niveles de servicios de una red. Luego, se espera construir un monitor activo de sistemas distribuídos, que permita tanto medir como proyectar el desempeño de éstos mediante la incorporación de los datos de mediciones a un modelo de red de espera.

IV.- CONCLUSIONES

Mediante este trabajo se han construído herramientas computacionales que facilitaron el desarrollo de un sistema de acceso transparente a recursos distribuídos en una red local universitaria, incluyendo desde simples comandos hasta aplicaciones diversas. De paso, se han incorporado herramientas que permiten obtener el desempeño de este tipo de sistemas.

La interfaz de comunicación desarrollada permite ocultar los detalles de bajo nivel de la red, haciéndola transparente tanto al programador de sistemas distribuídos

como a los usuarios. Permite separar los aspectos de comunicación de la problemática inherente a la aplicación propiamente tal.

La interfaz de usuario disponible a nivel de PCs, permite que el usuario vea los recursos remotos como si residieran en su PC. Además, permite el uso concurrente de varias aplicaciones remotas. Esto se logra haciendo que cada aplicación despliegue su salida en una ventana independiente del micro-computador.

BIBLIOGRAFIA

[Aur90] Aurtenechea F, Wallem A (1990) Una Herramienta para el Desarrollo de Sistemas Distribuídos en una Red Heterogénea. **XVI Conferencia Latinoamericana de Informática**, Asunción, Paraguay, 1066-1080, Sept.

[Che88] Cheriton D. (1988) The V Distributed System. **Communicationof the ACM**, March, Vol 31, No. 3, 314-333.

[Exc88] Excelan Inc. (1988) Socket Library Application **Program Interface User's Guide.**

[Gla89] Glass L. (1989) "Building Heterogeneous Networks". **Byte,** (Sept. 1989).

[Iso81] Iso (1981) Open System Interconnection, Basic Reference Model, (ISOdraft proposal 7498) **ISO TC97/SC16. Computer Communications,** 15-65

[Met76] Metcalfe R., Boggs R. Ethernet: Distributed Packet Switching for Local Computer Networks. **Communicaction of teh ACM**, Vol. 19, Nº 7, 395-404.

[She86] Shelzer, A., Popek G.(1986) Internet Locus: Extending transparency to an Internet environment, **IEEE Transactions on software Engineering**, Vol SE-12, No 11, 1067-1075.

[Tan85] Tanenbaum, A. y Van Renesse, R. (1985) Distributed Operating Systems. **Computing Surveys**, Vol. 17, no. 4, 419-470

[Tan88] Tanenbaum, A. (1988) Computer Networks. Prentice Hall, New Jersey.

INTELIGENCIA ARTIFICIAL E O CONTROLE DE PROCESSOS*

Mauricio G. Solar F.[1/2] Sueli B. T. Mendes[1]

[1]COPPE/UFRJ

Programa de Engenharia de Sistemas e Computação

Centro de Tecnologia - COPPE - UFRJ - Caixa Postal 68511

Rio de Janeiro - RJ - CEP 21945 - Brasil

Endereço Eletrônico: cos99288@UFRJ (BITNET)

[2]USACH

Universidad de Santiago de Chile

Departamento de Ingenieria Informática - Casilla 10233 - STGO

Avda. Ecuador 3659 - Estación Central - Santiago de Chile

FAX: (562) 6811422

RESUMO

O trabalho apresenta a implementação do hardware e do software de um Sistema Especialista (SE) numa aplicação de automação e controle de processos. Detalha-se o esquema do software (mecanismos de inferências e representação do conhecimento) e do hardware sob o qual foi desenvolvido o SE.

O sistema especialista desenvolvido têm como objetivos melhorar a qualidade, segurança e eficiencia dos processos controlados, o que diminui o desgaste físico dos operadores, portanto são diminuidos os riscos de falhas humanas.

* Este trabalho foi parcialmente financiado pelo Centro de Pesquisas de Energia Elétrica (CEPEL-Rio) e pelo CNPq.
CEPEL: Depto. de Eletrônica - Caixa Postal 2754
Rio de Janeiro - RJ - CEP 21941 - Brasil

1. INTRODUÇÃO

Os pesquisadores da área de IA [1] indicam a existência de quatro classes de Sistemas Especialistas (SE) [2]:

1.- **Sistemas Consultores:** onde a entrada é dada pelo homem e a saída do sistema é para o homem;

2.- **Controladores Inteligentes:** entrada do sistema é dada pelo homem e a saída é através de sensores (encontrado nas indústrias automatizadas);

3.- **Classificação Inteligente de Dados de Sensores:** entrada dada por sensores e a saída do sistema é para o homem;

4.- **Dispositivos Realmente Autónomos:** entrada e saída através de sensores, onde as ações são baseadas nos dados dos sensores.

O uso de técnicas heuristicas em **Controladores Inteligentes** [3] têm um duplo objetivo:

- o controle supervisor tem um razocinio heuristico para tomar decisões dos resultados obtidos por métodos heuristicos ou analíticos;
- o controle direto interpreta os dados dos sensores para tomar decisões de controle.

A solução adoptada para implementar o SE para controlar processos em tempo real é um **Controlador Inteligente**. Este Controlador necessita de uma entrada, que é dada por outro SE em configuração de processos [4]. A entrada consiste de um arquivo de comandos (Base de Dados, BD) que contêm as instruções que devem ser executadas durante o processo.

Este Software de automação de processos efetua três tarefas básicas: **aquisição, análise** e **presentação** dos dados do processo.

Para a realização destas tarefas foi implementada uma Interface Homem-Máquina (IHM) com a qual o operador pode interagir para acompanhar o processo [5]. Este sistema interpreta e executa as operações lidas da BD de entrada. A BD contêm o código em alto nível das operações que devem ser efetuadas na execução do processo. Cada operação em alto nível é traduzida em comandos de baixo nível que são enviados para os equipamentos climáticos ou para os instrumentos de medição. Estes comandos consistem na configuração dos equipamentos climáticos e instrumentos de medição. Os equipamentos climáticos são os encarregados de manter as condições climáticas de temperatura, pressão e umidade do processo para permitir o controle. Os instrumentos envolvidos em cada etapa permitem a monitoração dos sinais de interesse.

No final da execução tem-se toda a informação referente à evolução do processo armazenadas em diferentes BDs, tais como monitoração dos equipamentos (temperatura, pressão e umidade), monitoração dos instrumentos (tensão, corrente, frequência, etc.), alarmes ocorridos e comandos executados.

Todo o controle do processo é efetuado em base ao conhecimiento representado [6], armazenado e manipulado [7] pelo SE.

Neste trabalho detalha-se o projeto de Automação de Testes de Equipamentos (ATE) desenvolvido e implementado no Laboratório de

Ensaios Climáticos do Centro de Pesquisas de Energia Eletrica (CEPEL-Rio de Janeiro). A automação do sistema tem como objetivo desenvolver um sistema inteligente que auxilie o operador na supervisão, execução e controle dos Ensaios Climáticos (EC). O sistema de operação mostrado neste trabalho foi desenvolvido como um primeiro estágio do projeto para se obter uma base sob a qual avaliar o desempenho do sistema para continuar evoluindo incrementalmente com o desenvolvimento do sistema até integrar todos os equipamentos e instrumentos do laboratório.

2. AUTOMAÇÃO DOS ENSAIOS CLIMÁTICOS (EC)

O EC tem basicamente três fases principais: Configuração, Operação e Avaliação. A fase de Configuração tem como objetivo principal a especificação total do EC, no qual o especialista estuda e seleciona as normas que melhor se adaptam ao teste, segundo as características de funcionamento do objeto sob teste. Uma vez escolhidas as normas, deve-se selecionar os equipamentos climáticos e instrumentos de medição que sejam mais adequados ao teste. Em seguida é necessário configurar os parâmetros dos equipamentos climáticos e dos instrumentos envolvidos em cada etapa do ensaio, bem como a sequência e a característica destas etapas [5].

Uma vez especificado o EC com todas as suas etapas, é possível iniciar a segunda fase. A fase de Operação tem como objetivo principal adquirir os dados, acompanhar e gerenciar o desenvolvimento do EC, no qual deve-se monitorar, supervisionar e

controlar o processo, que normalmente durará alguns dias, mas podendo durar até semanas.

A terceira e última fase é a Avaliação do objeto testado. Nesta fase o objetivo é avaliar o comportamento do objeto durante o EC, verificando se os dados adquiridos estão dentro da faixa de aceitação das especificações do objeto.

O projeto de Automação de Testes de Equipamentos engloba as três fases envolvidas num EC, ou seja, Configuração, Operação e Avaliação. O objetivo principal a ser alcançado com a ATE é diminuir, tanto quanto possível, a "interferência física do homem" dentro de um sistema totalmente desenvolvido e integrado, onde a atuação humana passa a ser mais de supervisão.

A automação da fase de Operação dos EC's foi desenvolvida como um primeiro estágio da automação total do sistema. O "hardware" envolvido neste sistema é mostrado na figura 1, cuja descrição detalhada se encontra em [8].

2.1. Automação da Fase de Operação

O sistema de automação pode ser visto, em linhas gerais, como um sistema de supervisão e controle em tempo real que manipula uma grande quantidade de informações. O "software" envolvido na fase de Operação é estruturado em módulos que se relacionam em diversos níveis hierárquicos. O nível hierarquicamente mais alto é aquele onde todo o "hardware" do sistema é transparente para o

usuário. O nível mais baixo lida com as instruções que atuam diretamente sobre o "hardware" do sistema.

Este "software" foi organizado em três subsistemas independentes:

.**Subsistema Gerente:** gerenciador dos processos. Encarregado de ler os comandos, interpretá-los e enviá-los aos outros subsistemas através de um módulo chamado **Identificador;**

.**Subsistema TAC** (Terminal de Aquisição e Controle): usado como interface com os equipamentos. O TAC permite controlar de forma independente os equipamentos condicionadores climáticos [9];

.**Subsistema GPIB** (General Purpose Interface Bus): usado como interface com os instrumentos. Esses instrumentos respondem ao padrão GPIB especificado pela Hewlett-Packard, também conhecido como barramento IEEE-488, amplamente usado na indústria.

Os três subsistemas (figura 2) estão ligados por um módulo de interface denominado módulo **Identificador**. A descrição detalhada deste módulo, bem como dos módulos TAC e GPIB, se encontram em [8].

2.2. Subsistema Gerente

O nível hierarquicamente mais alto corresponde ao subsistema Gerente, o qual é composto por uma IHM e um Gerenciador do Processo. A IHM permite ao operador acompanhar o andamento do processo, como, por exemplo, observar medidas obtidas, comandos executados, gráficos de acompanhamento, alarmes ocorridos, etc. Todas as informações apresentadas pela IHM são obtidas das

diferentes BD's, as quais são atualizadas em tempo real, ou seja, o operador do sistema sempre tem disponíveis as últimas informações do processo.

O nível seguinte diretamente relacionado com a IHM corresponde ao Gerenciador do Processo. O Gerenciador é quem determina, a partir da BD chamada Agenda, que contém a configuração completa do ensaio, as operações que devem ser efetuadas no andamento do processo.

A função do Gerenciador é:

- interpretar os comandos de configuração;
- servir de interface entre o Gerente e os subsistemas GPIB e TAC, construindo, a partir desses comandos, as mensagens a serem enviadas a estes subsistemas e recebendo as respectivas mensagens retornadas;
- gerenciar as BD's que armazenam as informações obtidas dos subsistemas.

Intérprete de Comandos

A interpretação dos comandos consiste em três fases: ler um comando da BD da Configuração (Agenda), verificar se não contém erros e preparar a mensagem para enviar aos subsistemas TAC ou GPIB. Estes comandos podem ser do tipo:

- Configurar um instrumento para fazer uma medida;
- Configurar um equipamento climático para atingir uma temperatura determinada;

- Iniciar uma operação no equipamento climático;
- Cancelar uma operação no equipamento climático;
- Perguntar pelo status atual em um equipamento climático;
- Perguntar qual o valor de uma medida em um instrumento;
- Indicar as ligações entre os instrumentos que se deve efetuar.

Interface com o Processo

A Interface com o Processo constrói uma estrutura de dados para ser enviada aos subsistemas TAC ou GPIB. As mensagens a serem enviadas têm uma estrutura única que identifica para qual subsistema elas se destinam, indicando o instrumento ou equipamento correspondente, com os dados que são necessários. Esta estrutura contém as seguintes informações:

- Identificador GPIB ou TAC;
- Identificação de instrumento ou equipamento;
- Tipo de operação que se deseja efetuar;
- Parâmetros da operação a realizar no instrumento ou equipamento;
- Início, duração e/ou término da operação.

Gerenciador de Bases de Dados (GBD)

A IHM exterioriza as informações provenientes das BD's, as quais são preenchidas dinamicamente em tempo real pelo GBD. O GBD se comunica com os dois subsistemas de aquisição de dados (TAC e GPIB) através da Interface com o Processo.

O subsistema TAC é usado para monitorar e controlar os equipamentos envolvidos no processo e o subsistema GPIB é usado como interface dos instrumentos com o sistema.

A cada vez que é enviada uma mensagem para qualquer um dos subsistemas, é retornada uma mensagem, na qual se tem informação a respeito do andamento do processo. O GBD armazena esta informação na BD correspondente. Existem quatro BD's, cada uma contendo informações específicas sobre o processo, que são as seguintes:

- ALARMES: armazena todos os alarmes ocorridos no transcurso do processo;
- TAC: armazena as medidas relativas aos equipamentos;
- IEEE: armazena as medidas obtidas dos instrumentos;
- ROTEIRO: armazena os comandos executados no processo.

2.3. Módulo Identificador

O módulo Gerente da Operação interpreta instruções de operações lidas de uma BD de entrada e as executa. Estas instruções contidas na BD são códigos em alto nível das operações a serem realizadas no ensaio. Cada operação codificada em alto nível é traduzida em comandos de baixo nível que são enviados para os subsistemas TAC ou GPIB. Estes comandos constituem a configuração dos equipamentos e dos instrumentos envolvidos em cada etapa, para permitir a monitoração e controle do EC.

O módulo Identificador recebe a mensagem enviada pelo Gerente e identifica se o comando de operação ou instrução pertence

ao subsistema GPIB ou TAC, passando o comando para o subsistema correspondente.

2.4. Subsistemas TAC e GPIB

O nível hierárquico imediatamente inferior ao do módulo Identificador contém os subsistemas GPIB e TAC, que determinam, a partir da mensagem recebida do módulo Identificador, o equipamento ou instrumento ao qual se destina o comando.

O subsistema TAC, coordenado pelo subsistema Gerente, é responsável, basicamente, pelo controle dos equipamentos condicionadores climáticos. Neste subsistema estão armazenados em forma distribuida os algoritmos de controle de cada equipamento.

O desenvolvimento deste sistema de automação está sendo realizado em etapas. Primeiramente, foi implementada a parte correspondente à automação de um único equipamento condicionador climático, no caso, uma Estufa. Atualmente está sendo desenvolvida a parte correspondente à automação de uma Câmara Climática, e assim progressivamente, até a integração completa de todos os equipamentos do laboratório.

O subsistema GPIB é o responsável pela recepção das mensagens destinadas aos instrumentos ligados ao barramento GPIB, pela identificação do instrumento ao qual a mensagem se destina, pelo envio das mensagens aos módulos correspondentes e pela recepção das respectivas respostas, encaminhando-as ao módulo

Identificador. Nesta fase do projeto estão sendo utilizados três instrumentos: um Osciloscópio, um Multímetro e um Sintetizador.

Cada instrumento e equipamento tem características próprias e, para fazer a interface adequada, existe um módulo de "software" correspondente a cada um, conforme indicado na figura 2 (instrumento 1, instrumento 2, ..., equipamento 1, equipamento 2, ...). Estes módulos recebem as instruções e as coloca em forma compatível com os parâmetros de cada instrumento ou equipamento, sendo este o nível hierarquicamente mais baixo.

3. POR QUE INTELIGÊNCIA ARTIFICIAL ?

O sistema de automação descrito neste trabalho é uma ferramenta de auxílio disponível para o especialista que lhe proporciona facilidades para verificação do andamento do processo.

No desenvolvimento da fase de Operação existe a necessidade de um grande conhecimento dos alarmes que podem ocorrer na execução de um processo. Além deste conhecimento, também é necessário conhecer as decisões que devem ser tomadas, com a finalidade, por exemplo, de analisar o alarme, o estado do processo e a condição em que está operando (onde se supõe existir algum problema), para efetuar uma determinada ação. Este processamento de alarmes é um conhecimento armazenado na forma de regras (SE ... ENTÃO ...).

Outro conhecimento especializado que deve estar presente nesta fase corresponde às características dos equipamentos

condicionadores climáticos e às características dos instrumentos usados para a aquisição de dados. Este conhecimento, pelo fato de ser bem estruturado, foi representado na forma de frames [6].

Detecta-se aqui a existência de um domínio de conhecimento muito raro e de alto custo. Raro, pelo fato de que os especialistas humanos são escassos e, quando disponíveis, têm pouco tempo livre, dado que cada ensaio tem uma grande duração no tempo. Embora os especialistas sejam muito eficientes em seu trabalho, a possibilidade de confusão causada por muitas solicitações conflitantes os torna mais vulneráveis a erros do que um sistema baseado em computador. O alto custo é uma questão de tecnologia, dado que, com o advento dos microcomputadores pessoais, é possível implementar SE's com um custo relativamente baixo.

Com a ATE pretende-se atingir o objetivo de diminuir os erros no controle de um ensaio, deixando ao especialista mais tempo para outras tarefas, evitando erros que podem invalidar um ensaio depois de ter sido executado durante algumas semanas.

4. COMENTÁRIOS E CONCLUSÕES

O SE foi desenvolvido sob o sistema operacional com primitivas em tempo real, chamado Núcleo do Sistema Operacional (NSO), desenvolvido pelo CEPEL para propósitos de controle [9]. O nível hierarquicamente mais alto na estrutura do software, o subsistema Gerente, foi desenvolvido em Turbo Prolog [10], para o qual foi necessário desenvolver uma interface em assembler 8086

para permitir a comunicação com a linguagem PLM-86, com a qual foram desenvolvidos os níveis mais baixos na hierarquia do software, os subsistemas TAC, GPIB e os módulos dos instrumentos e equipamentos.

A linguagem Turbo Prolog foi escolhido pela eficiencia e velocidade na execução de programas em Prolog [11].

O PLM-86 foi escolhido por ser uma linguagem de alto nível com facilidades de baixo nível e possuir uma grande portabilidade entre os produtos da Intel.

Uma grande vantagem, importante de mencioanr, é a facilidade de integrar novos instrumentos e novos equipamentos ao sistema. Esta facilidade deve-se principalmente à forma na qual foi definido a base de conhecimento. Este conhecimento está representado na forma de frames, e todas as operações sobre esses frames está disponível dentro do sistema.

O SE implementado permitiu testar através de alguns Ensaios Climáticos, o bom desempenho no controle de processos em tempo real. O sistema consiguiu controlar, monitorar e supervisionar eficientemente durante vários dias alguns EC's. Este SE respondeu satisfatoriamente às necessidades do projeto do CEPEL. Nesta afirmação está inserida uma grande implicação: a viabilidade da aplicação de técnicas de Inteligência Artificial no controle de processos.

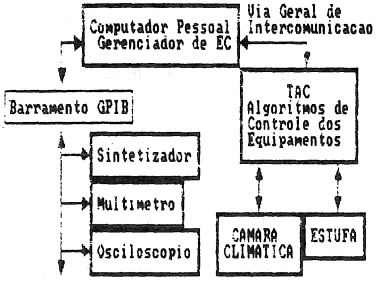


Figura 1: Hardware envolvido na ATE.

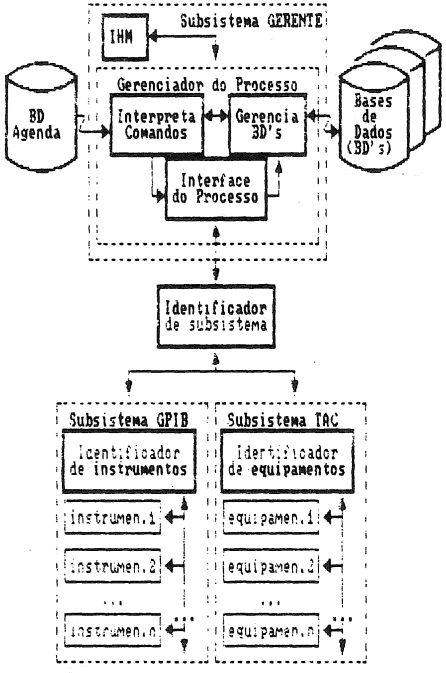


Figura 2: Esquema do Software da fase de Operacao.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] LEINWEBER, D., "Expert Systems in Space", IEEE Expert, 1987.

[2] HAYES-ROTH, F., D. WATERMAN & D. LENAT, Building Expert Systems, vol.1, Addison-Wesley, London, 1983.

[3] ASTROM, K.J., "Toward Intelligent Control", IEEE Control Systems Magazine, pp 60-64, April 1989.

[4] SOLAR F., M., O. APPEL, M. AMORIM & SUELI T. MENDES, "Um Sistema Especialista para Configurar Testes e Ensaios Climáticos", 6º Simpósio Brasileiro em IA, Rio de Janeiro, Novembro 1989.

[5] SOLAR F.,M., Metodologia de Desenvolvimento de um Configurador de Testes Automáticos com Técnicas de Inteligência Artificial, Tese de M. Sc. da COPPE/UFRJ, Agosto 1989.

[6] MINSKY, M., A framework for representing knowledge, in The Psychology of Computer Vision, McGraw-Hill, 1975.

[7] KELLER, R., Expert System Technology: Development & Application, Prentice-Hall, New Jersey, 1987.

[8] ALBUQUERQUE, A., E. MENDONÇA, Sistema de Automação do Laboratório de Ensaios Climáticos do CEPEL, Escola de Engenharia, Depto de Eletrônica, UFRJ-RJ, Janeiro 1989.

[9] Centro de Pesquisas de Energia Elétrica (CEPEL), Sistemas Digitais Distribuídos para a Automação de Usinas e Subestações, Vol. I - Manual do Hardware, 1988.

[10] BORLAND INTERNATIONAL, Turbo Prolog ver. 2.0 User's Guide, 1988.

[11] WONG, W., "Prolog: A Language for Artificial Intelligence", PC Magazine, October 1986.

SIEN:ES UN "SHELL" DE SISTEMAS EXPERTOS MULTIPLES QUE RECIBE UN TEXTO DE ENTRADA

MANUEL GONZALEZ HERNANDEZ
FRANCISCO GALAN JIMENEZ
JESUS GONZALEZ CUEVAS

INSTITUTO POLITECNICO NACIONAL
Centro Nacional de Cálculo
Av.IPN, s/n Col. Lindavista CP. 07300
México, D. F. Tel. 586-4711 X 44

RESUMEN

El artículo trata con el manejo de información de un texto que un usuario del Sistema Experto Nodriza (SIEN) suministra para explicar su problema. De este texto es posible seleccionar una base de conocimientos y clasificar los hechos (si los hay) para distribuirlos en las bases de conocimiento seleccionadas con el fin de que el sistema pueda concluir inmediatamente. La idea principal del trabajo es mostrar los algoritmos para seleccionar una o varias bases de conocimientos a las que se les puede anexar hechos que provienen del texto del usuario para resolver el problema en mano. En sistemas expertos múltiples como SIEN [González, H. M. 1990 h] se requiere la solución de este problema para compartir información entre bases de conocimiento y hacer posible la comunicación de sistemas expertos.

Palabras clave: Sistemas Expertos/ Sistemas Expertos Múltiples/ Sistemas Basados en Concimiento/ Bases de Concimiento.

INTRODUCCION

El propósito de este artículo es mostrar algunos resultados de los algoritmos para seleccionar una base de conocimiento y clasificar hechos de un texto que un usuario escribe para explicar su problema.

Las nuevas tendencias en Sistemas Expertos (SE) se manifiestan por la investigación en el diseño y construcción de Sistemas Expertos Múltiples (SEM) [Silverman, B. G. 1988a, 1989b], [LeClair, S. R. 1989], [Wolf, W. A. 1989], [González H. M. 1988,1989,1990h], en donde se pretende manejar varias bases de conocimiento (BC) y comunicar sus resultados. Básicamente se quiere encontrar soluciones de problemas con una cierta mezcla de información proveniente de diferentes bases de conocimiento, de tal manera que sea posible construir SE con BC pequeñas con algún grado de comunicación. Por otra parte también se requiere que el usuario pueda interaccionar con la computadora en una manera semejante al sistema de consulta que se forma entre un experto humano y un cliente. Es decir: la idea es crear un ambiente de programación similar al sistema de consulta. Para lograrlo supondremos que el usuario puede escribir un texto con la información conveniente para la consulta con el sistema, así que el sistema experto múltiple pueda seleccionar una base de conocimiento y además obtener de este texto los datos (hechos) factibles para la solución del problema (si los hay).

En el Sistema Experto Múltiple SIEN [González ,H. M. 1990 h], las bases de conocimiento tienen asociado un grafo dirigido llamado el inventario. El inventario es un conjunto de parejas de palabras clave que representran una relación binaria entre los elementos de la base.

El SIstema Experto Nodriza (SIEN) [González, H. M. 1986b,1987c 1988d] recibe este nombre por que tiene facilidades de manejar otras BC sin afectar su estructura, por que cuenta con mecanismos de inferencia que interpretan el conocimiento dependiendo del esquema de representación usado y porque la BC insertada puede comunicarse con las ya registradas por SIEN. En la figura 1 se muestra un diagrama de la arquitectura del sistema.

La idea original de la construcción del sistema SIEN fué, que a través de las evidencias del usuario el sistema se pudiera situar en alguna parte de la base de conocimiento en relación a lo que decía el usuario en un texto y desde ahí iniciar la consulta. Esto se logró considerando que la base de conocimiento se podía dividir en partes que se llamarían temas [González H. M. 1989 e, 1989 f]. Una vez dividida la base de conocimiento el problema de colocarse en uno de los temas resultaría más sencillo, para lo cual se propusó que cada tema tuviera asociado un inventario. El inventario constaría de palabras características del tema, por ejemplo, la suspensión de un carro es un tema, que se puede identificar cuando se habla de "los amortiguadores", como palabra clave. Cuando se habla de un tema siempre se introducen palabras características que lo distinguen de otros. Por ejemplo, si se

habla de alguna clase de peces o de alguna parte de la bicicleta, el uso de palabras claves nos ubican en ellos. Se puede pensar entonces en bases de conocimiento genèricas [Chandrasekaran, B. 1984].

ESQUEMA GENERAL DEL SIEN

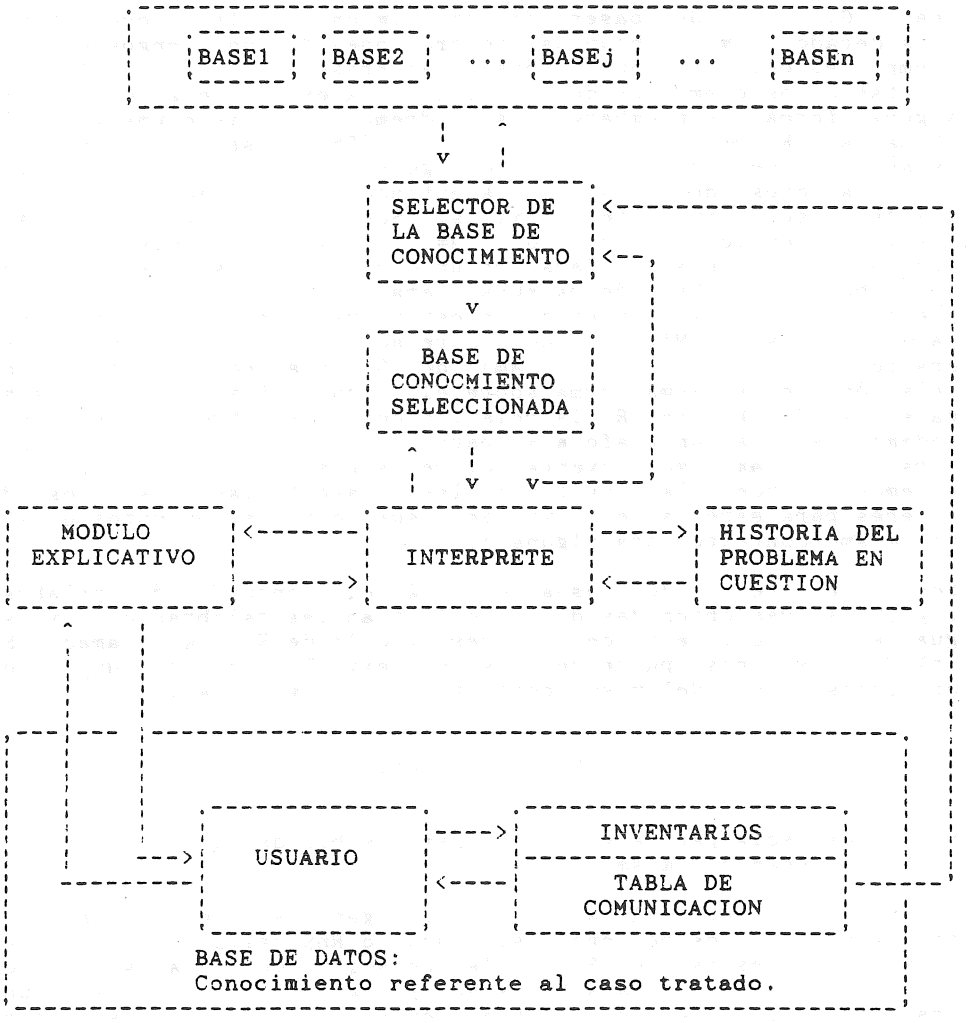


Figura 1.

La primera versión de SIEN [González, H. M. 1988a] fué escrita en MicroProlog [Clark, MacCabe 1984], con un solo esqueleto (modelo de MYCIN) y no puede ejecutar otros intérpretes escritos en otros lenguajes de programación porque la versión de Microprolog usada no es un compilador. Actualmete se cuenta con una versión escrita en lenguaje de programación C [Galán, J. F. 1990], que si puede ejecutar otros sistemas expertos aún escritos en otros lenguajes.

SELECCION DE BASES DE CONOCIMIENTO

Sean G1,G2,...,Gk bases de conocimiento, las que hemos considerado como conjuntos de oraciones de un lenguaje, por ejemplo el cálculo proposicional [Van Frassen G., M., 1971]. Por lo tanto los elementos de Gj j=1,2,...,k estan relacionados en alguna forma, sin embargo, supondremos que existe una relación (binaria) [Kolman,B. & Busby, R.. C. 1986 y Ross,K. A. 1990] de estas oraciones (posiblemente de algunas), es decir, que si A y B son oraciones de Gj j=1,2,...k entonces A se relaciona con B a través de sus componentes atómicos. Sean, A1j, A2j, ..., Amj oraciones en los conjuntos Gj, j=1,2,...,k, (que distinguen a cada Gj de las demás). Cada conjunto de oraciones {Amk} contiene un subconjunto {Psj} de palabras características de cada Gj que pueden definir su contexto, a estas palabras les llamaremos palabras clave. Nótese que la relación (binaria) entre las oraciones A1j, A2j, ..., Amj de Gj para cada j induce una relación de la misma forma sobre el conjunto Kj={Psj} con s<=m para j=1,2,..k. Sea Rj la relación definida sobre Kj, así que podemos asociar un grafo a Rj para j=1,2,...,k. A este conjunto (grafo) le llamaremos inventario. De la gráfica de Rj j=1,2,...,k podemos conocer la longitud de las trayectorias, ésto es de interés para saber si el texto del usuario y los conjuntos Gj son del mismo contexto para alguna j.

Sea T el texto del usuario y Q el conjunto de palabras significativas obtenidas de T (se filtran las palabras de T y se guardan en Q), entonces la restricción de Rj a Q llamada Sj j=1,2,...,k, nos puede decir si el texto T y los conjuntos de oraciones Gj son del mismo contexto. (^, significa intersección)

$$Sj = Rj \hat{} (QxQ)$$

Si Sj es vacía para alguna j no hay posibilidad de que para esa j, T y Gj sean del mismo contexto.

Si Sj es distinto del vacío se investigan sus elementos para ver si alguno de ellos pertenece al conjunto Rnj j=1,2...,k donde Rnj (potencia n-ésima de Rj) es la relación definida sobre los conjuntos Kj para cada j. x Rnj y significa que existe una trayectoria de longitud n de x a y en Rj j=1,2,...,k. También se puede pedir que los elementos pertenezcan al conjunto Rooj, donde Rooj es una relación definida sobre los conjuntos Kj para cada j. x Rooj y significa que existe alguna trayectoria en Rj j=1,2,...,k de x a y. En este caso excluimos todas las

trayectorias de longitud uno ya que no serian muy significativas para saber si T y Gj son del mismo contexto, (Nótese que Rooj = R1j U R2j U ... U Rnj).

De esta forma seleccionamos cuales de las Gj son factibles para el tratamiento del problema planteado por el usuario en el texto.

Una vez determinadas todas las Gj factibles para la solución del problema se jerarquizan de acuerdo a las longitudes de las trayectorias, ésto se hace contando los elementos de Sj en Rooj, (se pueden jerarquizar de alguna otra manera). Así, se forma una lista de bases de conocimiento Gj para algunas j's (posiblemente todas o ninguna) y se escoje la primera (en el caso de que la lista no sea vacía) para reconfigurar un sistema experto. El sistema SIEN busca el nombre de la base en una tabla de registro para saber que tipo de intérprete le corresponde y que parámetros requiere para reconfigurar un sistema experto relativo a la base de conocimiento Gj. De esta manera, una vez instalado el sistema digamos X1 toma una regla de la Gj en cuestión y la ejecuta, si su valor es verdad verifica en una tabla de comunicación [González 1989f 1989g] y si la encuentra digamos que es Gm anuncia la comunicación y el usuario decide si cambia de experto o no (con la nueva Gm). Si el caso es afirmativo guarda el estado de X1 e instala un nuevo X2 con la Gm comunicada de esta forma se van formado cadenas de expertos, los cuales pueden explicar su línea de razonamiento global o particularmente. Si X1 se mantiene hasta agotar sus reglas o bien no hay comunicación, SIEN toma el siguiente elemento de la lista de las G's y repite el proceso, sin embargo, X2 puede llamar a otro experto digamos Xt. Esto depende de como esta construida la tabla de comunicación [González 1989g].

Resumiendo los pasos del algoritmo tenemos,

1.- Se da un texto y se construye el conjunto Q.
2.- Se construye una relación Rj inducida por la relación de los elementos de Gj para j=1,2,...,k
3.- Se obtiene el conjunto Sj=Rj ^ (QxQ)
4.- Si Sj=vacío para toda j no hay selección de las Gj's, el proceso para y se regresa al paso 1
5.- Si Sj distinta del vacío, se construye Lj=Sj ^ (Rooj-Rj)
6.- Se encuentra la longitud de los elementos Lj.
7.- Si los elementos de Lj son de longitud menor que el Max{Rooj}, el contexto es débil.
long
8.- Si los elementos de Lj son de longitud igual a Max{Rooj} el contexto es fuerte.
long
9.- Se construye la lista de los Gj de acuerdo al contexto débil o fuerte para jerarquizar.

El cálculo de Roo se puede hacer, a través del algoritmo Warshall, que se muestra a continuación.

Supóngase que R es una relación en un conjunto A. Una trayectoria de longitud n de a a b es una sucesión finita

$p = a, x_1, x_2, \ldots, x_{n-1}, b$, que se inicia con a y termina con b,

tal que

$$a\ R\ x_1,\ x_1\ R\ x_2,\ \ldots,\ x_{n-1}\ R\ b$$

Obsérvese que una trayectoria de extensión n involucra a n + 1 elementos, aunque no necesariamente distintos.

Una trayectoria se puede visualizar fácilmente con ayuda del grafo dirigido de la relación. Esta aparece como una "trayectoria" geométrica o una sucesión de aristas en el grafo dirigido, donde las direcciones indicadas en las aristas se siguen y de hecho una trayectoria lleva ese nombre por esta representación. Por consiguiente, la longitrud de una trayectoria es el número de aristas en la trayectoria, donde no es necesario que todos los vértices sean distintos.

Es claro que las trayectorias de longitud 1 pueden identificarse con los pares ordenados (x,y) que pertenezcan a R. Las trayectorias en una relación pueden usarse para definir nuevas trayectorias las cuales son muy útiles. Si n es un entero positivo fijo se define una relación R^n en A como sigue: x R^n y significa que existe una trayectoria de longitud n de x a y en R, También se puede definir una relación Roo en A, haciendo que x Roo y signifique que existe alguna trayectoria en R de x a y. La longitud de esta trayectoria dependerá, en general, de x, y de y. A Roo se le llama algunas veces relación de conectividad de R.

Sea R una relación en un conjunto finito $A = \{a_1, \ldots, a_n\}$, y sea $M_R = [m_{ij}]$ la matriz de n x n de R. Entonces $m_{ij} = 1$ si y sólo si $a_i\ R\ a_j$.

Teorema 1 Para n>=1, y R una relación en un conjunto finito A, se tiene

$$M_{R^n} = M_R \cdot M_R \cdot \ldots \cdot M_R \quad (n \text{ factores})$$

Teorema 2 Sea A un conjunto con n elementos, y sea R una relación en A. Entonces, $Roo = R^1 \cup R^2 \cup \ldots \cup R^n$

Las matrices asociadas a R^1, R^2, ..., R^n, son las potencias de M,

Del teorema anterior encontramos que las potencias de R mayores que n no son necesarias para calcular Roo. Esto implica calcular las potencias de M para encontrar Roo. Sin embargo, esto se puede evitar si aplicamos el algoritmo de Warshall.

ALGORITMO DE WARSHALL

Sea R una relación en un conjunto A= {a_1, a_2, ..., a_n}.
Se introduce una sucesión de matrices W_0, W_1, ..., W_n tal que
$W_0 = M_R$ y $W_n = M_{Roo}$, y cada matriz se calcula fácilmente a

partir de la anterior. Estas son distinas de las potencias de M, y esta diferencia resulta de gran ayuda en el cálculo. En una trayectoria x_1, x_2, ..., x_n, cualquier vértice excepto el primero y el último (en este caso x_1, x_2, ..., x_{n-1}) se llama vértice interior. Para cualquier entero fijo k entre 1 y n, s define W como la matriz que tiene un 1 en la psociión i,j si y sólo si existe una trayectoria en R de a a a cuyos vértices interiores, si existen, vienen del conjunto {a_1, a_2,..., a_k}. En otras palabras, los vértices interiores no pueden tener subíndices mayores que k. Se define W_0 como M_R. Se ve que $W_n = M_R$ ya que cualquier trayectoria deberá tener vértices interiores que vengan de {a_1, a_2, ... a_n} = A. Ahora se mostratá cómo se calcula cada W a partir de las matrices W_1, W_2, ..., W_{k-1}
Sea $W_k = [w_{ij}]$, esto es, w_{ij} es el componente en la posición i,j de la matriz W_k. Entonces se tiene el siguiente resultado.

Teorema 3 Para cualquier i,j, k $0 \leq i,j, k \leq n$.

$$w_{ij}^{(k)} = w_{ij}^{(k-1)} \vee (w_{ik}^{(k-1)} \wedge w_{kj}^{(k-1)})$$

EJEMPLO

Como un ejemplo se consideraron dos bases de conocimiento, un conjunto de palabras clave pora cada una de ellas, su grafo correspondiente según la relación que existe por dichas palabras en cada base y un texto fabricado exprofeso para ilustrar el algoritmo.

Observése que la selección de las palabras clave es arbitrario y que son sugeridas por el experto humano en la construcción de la base, así como las ligas que existen entre reglas en función de estas palabras.

En la determinación de la relación entre reglas se considera, aunque no es única, que los átomos consecuentes que contengan palabras clave serán inicio de una flecha en el grafo y su destino aquella regla cuyo átomo contiene una palabra clave que se encuentre como parte del antecedente.

Sean G1 y G2 dos bases de conocimientos dadas en la siguiente forma,

G1={r1,r2,r3,r4,r5,r6} y

G2={q1,q2,q3,q4,q5,q6,q7} en donde,

r1: IF A1(a,u,f) THEN C1(d)
r2: IF A1(a,u,f) THEN C2(h)
r3: IF A1(a,u,f) /\ C1(d) THEN C3(e)
r4: IF A3(i,j) THEN Co(d,e)
r5: IF A2(b,c) /\ C1(d) THEN C4(c,f)
r6: IF A1(a,u,f) /\ C4(c,f) THEN C5(h,d)

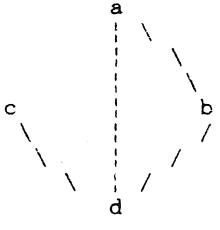
Cuyas palabras clave son

{a, b, c, d}

Por la relación que existe entre las reglas obtenemos el siguiente grafo que será considerado como el inventario de G1.

Oraciones relacionadas de acuerdo a las palabras clave: {(r1,r3),(r1,r5),(r5,r6)} y la relación de las palabras clave para formar su grafo es,

R1 = {(d,a),(d,c),(d,b),(b,a)



El conjunto G2 contiene las palabras clave {1,3,5,7} y las siguientes oraciones,

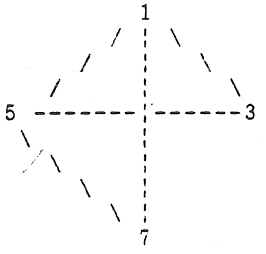
q1: IF B1(6,2) THEN D1(3)
q2: IF B2(6,7) THEN D2(4)
q3: IF B1(6,2) /\ D1(3) /\ D8(1) THEN D3(5)
q4: IF B2(6,7) /\ D3(5) THEN D4(6)
q5: IF D4(6) /\ B1(6,2) THEN D6(7)
q6: IF B3(3,5) /\ D1(3) THEN D7(3)
q7: IF B2(6,7) /\ D4(6) THEN D8(1)

Oraciones relacionadas de acuerdo a las palabras clave

{(q1,q6),(q1,q3),(q3,q4),(q3,q7),(q4,q5),(q4,q7)}

La relación de palabras clave para formar el grafo es,

R2 = {(3,5),(3,1),(5,1),(5,7),(7,1)}



Supongamos ahora que de un texto se obtuvieron las siguientes palabras significativas Q = {a s d}, entonces QxQ = {(a,a),(a,s), (a,d),(a,1), ... , (s,a),(s,s),(s,d),(s,1),..., (7,a),....,(7,7)} entonces siguiendo el algoritmo S1 = {(d,a)} y S2 = {(3,1)}, por lo tanto,

L1 = S1 ^ (Roo-R1) = {(d,a)}
L2 = S2 ^ (Roo-R2) = {(3,1)}

De esta forma sabemos que según el texto del usuario G1 y G2 han sido seleccionadas, ya que los elementos de L1, L2 tienen longitudes mayores que uno. Notamos que a través del texto podemos seleccionar varias bases a la vez. G1 y G2 se colocan en una lista como los nombres de los archivos que contienen las raglas antes descritas.

Los resultados que se obtuvieron fueron:

BUSQUEDA Y CLASIFICACION DE HECHOS

Supongamos que se tiene un texto T y sean G1, G2, ..., Gn bases de conocimiento, supongamos también que con el algoritmo de selección [González, H. M. 1990b] se han determinado (Gi, Gj, Gk) BC del texto T. Este algoritmo considera que cada BC tiene asociado un grafo dirigido llamado el inventario de la base,

constituido de palabras características o palabras clave que distingue a una BC de las demás. De esta manera el texto puede distinguir el contexto de la base y el sistema SIEN puede saber de que base se trata. Ahora nos interesa seleccionar las oraciones de T y clasificarlas para saber cuales son consideradas como hechos y cuales no. Estos hechos (si los hay) serán asignados a las BC Gi,Gj,Gk. Esto depende del tipo de esquema de representación del conocimiento empleado para las bases de conocimiento i,j,k. Por lo tanto debemos seleccionar las oraciones e investigar sobre el formato de los hechos de cada base (esto se debe hacer independientemente, es decir, para cada base), por lo que una vez identificada una oración se deberá transformar a hechos al formato del esquema de representación de la base (i,j,k). Esto es, con el fin de que el sistema experto SIEN no pregunte por los hechos que el usuario suministró en su texto de entrada y SIEN los utilice como información para alcanzar alguna meta. Así SIEN sólo preguntará por los datos faltantes sin alternativa.

Para poder establecer la transformación de un formato (lenguaje natural) a otro (esquema de representación de la BC) es necesario mencionar algunos elementos de los esquemas de representación.

REPRESENTACION DEL CONOCIMIENTO

La representación del conocimiento es muy versátil, es decir, existen algunas formas de representar conocimiento, pero no es simple hacer la seleción de entre todas las formas de representación. Generalmente la decisión está básada en la propia intuición o en las circunstancias particulares del problema en mano. Esto último tiende por sí mismo a una particular forma de representación. Este tipo de formas se les llama esquemas de representación y son conocidos como: Lógica, Reglas, Redes Semánticas, Marcos de Minsky, Objeto-Atributo-Valor (O-A-V), etc. De cualquier manera uno de los elementos que deben ser representados son los hechos. Los hechos son aseveraciones o afirmaciones de ciertos eventos de un fragmento de la realidad, así que estos elementos deberán ser representados y son precisamente nuestro interés.

Supongamos el esquema (O-A-V), usado en MYCIN [Shortliffe, E. H. 1976].

Cada hecho puede considerarse como una cadena de caracteres, por ejemplo:

ACERO-DUREZA-60=0.95

que se refiere a la oración o afirmación " El acero de grado 60 tiene una dureza de 0.95 "

La representación de este hecho en el esquema <O A V> se escribe como,

<ACERO> <DUREZA-60> <0.95>

Este esquema es el que utilza el Sistema Experto MYCIN [Buchanan, G. B. and Shortliffe, H. E. 1985].

De la misma manera podríamos escribir, por ejemplo, oraciones tales como:

"Es posible arrancar el carro" que transformada al esquema quedaría,

<CARRO> <ARRANCA> <POSIBLE>

Si la BC estructurada en reglas tiene un esquema de representación <O A V>, entonces, una regla general sería,

$$\text{IF } \bigwedge_{j}^{n} \langle O\ A\ V\rangle_j \text{ THEN } \langle O\ A\ V\rangle_k \quad \text{para } i<>k$$

entonces, en una red de inferencias las tripletas <O A V> corresponden a los nodos internos o nodos hojas (estos últimos son las ramas terminales de una red). En general los nodos hoja son nodos preguntables y por lo tanto estas tripletas se convertirán en hechos en el momento en el que el usuario los confirme. Estos hechos una vez confirmados pasarán a formar parte de la base de hechos.

Nuestra propuesta es, que de un texto que el usuario usa para explicar su problema, podamos extraer los hechos y usarlos en la consulta con el sistema experto formado [González, H. M. 1988a,1989a].

Para lograr nuestro propósito damos un texto en donde cada oración es separada con un carácter /. Se toma oración por oración y se transforma en tripletas <O A V>. Se comparan con aquéllas obtenidas de las oraciones establecidas en cada nodo preguntable de la red de inferencias. En este punto se está considerando las negaciones. Sin embargo, no se cuantifica la tripleta <O A V> con algún factor de certeza o alguna probabilidad. Lo cual podría resultar fácil ya que se le pediría al usuario que cuantificara cada oración con la probabilidad o el factor de certeza, encerrándolo entre paréntesis.

El siguiente procedimiento se ilustra con un ejemplo de una base ficticia de conocimiento relativa al desperfecto de autos.

PROCEDIMIENTO

El procedimiento mostrado es una alternativa que hemos escogido de algunas que pueden ser aplicadas para buscar y seleccionar los hechos que podrian estar contenidos en un texto. Una vez que los hechos son detectados, èstos deben ser clasificados y asignados a sus correspondientes bases de conocimiento como se dijo para las BC (i,j,k).

Supongamos entonces, que tenemos una BC con la representaciòn ya discutida, de ella formamos un conjunto de todos los nodos preguntables (es decir, sus oraciones asignadas a esos nodos). Estas oraciones se transforman en tripletas <O A V>, y con ellas formamos un conjunto digamos P. De la misma manera obtenemos un conjunto con las correspondientes tripletas de las oraciones del texto que llamamos T. Asi, si la intersecciòn entre T y P no es vacia quiere decir que el texto contiene hechos. Si la intersecciòn entre T y P es vacia, el usuario tiene la opciòn de dar otro texto o bien contestar las preguntas del sistema experto instalado. El procedimiento se aplica a cada BC, asi que los hechos son seleccionados para cada BC y por lo tanto el sistema experto instalado no preguntarà por estos hechos.

DESCRIPCION DEL PROCEDIMIENTO

El proceso supone creada la siguiente informaciòn

a) Un conjunto de palabras significativas en la base de conocimientos en la forma

palabra-natural o
palabra-natural [/sinònimo]...

las palabras se agrupan en tres secciones, terminadas con "@", que son: OBJETOS, ATRIBUTOS, VALORES Y NEGACIONES (y para la siguiente versiòn tambièn incluiremos probabilidades o factores de certeza)

b) Una base de conocimientos con oraciones de lenguaje natural, separadas por ':', identificados con el tipo de nodo 'LE' (hoja). (Como se muestra en un fragmento de la BC) La base deberà incluir el caracter "@" como terminador.

c) El texto del usuario en lenguaje natural, con oraciones separadas con "/". El texto debe terminar con "@".

Esta informaciòn deberà contener sòlamente caracteres vàlidos (caracteres ASCII con valor decimal en los siguientes rangos 32-122, 130, 160-173) ademàs del RETORNO y "~".

El procedimiento se desarrolló de la siguiente manera:

a) Lectura de las palabras-objetos y, en su caso, de los sinónimos asociados a éstas.

b) Lectura de las palabras-atributos y también, en caso de existir, de sus sinónimos.

c) Lectura de las palabras-valores y sus sinónimos.

d) Lectura de las palabras-negaciones y sus sinónimos.

e) Ordenamiento alfabético del vocabulario leído.

f) Lectura de la base de conocimientos, seleccionando sólamente las oraciones o nodos de nivel más bajo (es decir, las oraciones preguntables).

g) Conversión de las oraciones seleccionadas en (f) al esquema <O A V>. Para este propósito se llevó a cabo un barrido de cada oración buscando en el vocabulario todas las palabras significativas, al mismo tiempo que se sustituyeron por los sinónimos de mayor valor semántico.

h) Lectura del texto del usuario en lenguaje natural, identificando las oraciones por sus separadores ("/")

i) Conversión de las oraciones del texto a tripletas <O A V>. La conversión se realiza mediante un algoritmo semejante al usado en (g).

j) Detección de la intersección de los esquemas obtenidos en (g) y en (i).

RESULTADOS

Lo siguiente es un fragmento de una salida de la ejecución de uno de los algoritmos (el que clasifica los hechos).

```
A>caracas2
Introduzca el texto. Separe las oraciones con / y termine con @

Es difícil de aarrancar el carro/ Le falta potencia / Y la
máquina se sobre calienta/@

Las oraciones del texto:
0 Es difícil de arrancar el carro
1 Le falta potencia
2 Y la máquina se sobrecalienta
```

Los predicados O-A-V del texto son:

0	16 5 0 29	carro arranca dificil
1	0 61 0 38	potencia falta
2	53 67 0 69	máquina se sobrecalienta

Las oraciones preguntables de la base, ya confirmadas son:

1	16 5 0 29	carro arranca dificil
4	53 67 0 69	máquina se sobrecalienta

4. CONCLUSIONES

Se puede observar que los algoritmos nos ofrecen dos alternativas, la selección de la bases de conocimientos a través un texto que el usuario utiliza para explicar su problema y extraer hechos (si los hay). Esto es de beneficio, ya que se presentan otras alternativas en la investigación de sistemas expertos múltiples, sobre todo en aquellas aplicaciones, en donde el sistema experto recibe un mensaje o más especificamente en aquellos sistemas expertos de monitoreo.

AGRADECIMIENTOS

Este trabajo fué apoyado económicamente por el Sistema Nacional de Investigadores, la Comisión de Operación y Fomento de Actividades Académicas del I.P.N. (COFAA), la Comisión Nacional de Seguridad Nuclear y Salvaguardias y el Banco de México. También agradecemos el apoyo secretarial de la Srita. Patricia Rodríguez.

REFERENCIAS

Buchanan, G. B. and Shortliffe, H. E. (1985): Rule-Base Expert Systems, The MYCIN Experiments of Stanford Heuristic Programming Project, Addison-Wesley Publishing Company, USA.

Chandrasekaran, B., (1984): Expert systems:matching techniques to tasks. In W. Reitman (ed) Artificial Intelligence Aplications for Business. Norwood, Nj: Ablex.

Clark, McCabe., (1984): micro-PROLOG Programming in Logic, Prentice Hall International, N.J., USA.

Galán, J. F. (1990):SIEN es un "shell" de Sistemas Expertos Múltiples, Tesis de Maestria, Centro Nacional de Cálculo Instituto Politécnico Nacional, México D. F.

González, H., M.,(1988 a): SIEN: Prototipo de un sistema experto en Microprolog. Congreso Nacional de Computación. Pasado Presente y Futuro de la Computación en México. UNAM.

González, H., M., (1986 b): Skeleton for an expert system with knowledge base stratified in themes, Congreso Internacional del IEEE sección México, MEXICON 86, MEX040.

González, H., M., (1987 c): Estructura Básica de Un Sistema Experto de Multiesqueleto. Congreso Internacional del IEEE, sección México MEXICON87, MEX026.

González, H., M., (1988 d): Diseño de un Sistema Experto de Multiesqueleto. V Reunión Nacional de Inteligencia Artificial, Mérida Yuc. México. pp 207-220.

González, H., M., (1989 e): División de una Base de Conocimientos en Temas, VI Reunión Nacional de la Sociedad Mexicana de Inteligencia Artificial, Querétaro Qro. México, pp 431-451.
González, H., M., (1989 f): Extensión del Conocimiento Usando Sistemas Expertos de Multiesqueleto, Congreso Internacional del IEEE sección México, MEXICON89, MEX030.

González, H., M., (1989 g): SIEN: Es un Sistema Super Experto, Primer Congreso Nacional de Informática para la Ingeniería, Asociación Mexicana de Informática para la Ingeniería, Cuernavaca, Morelos.

González, H. M., (1990 h): Sistemas Expertos Múltiples, Segundo Congreso Iberoamericano de Inteligencia Artificial IBERAMIA 90, Morelia Mich. México.

Kolman, B. & Busby, R. C. (1986): Estructuras de Matemáticas Discretas para la Computación, Prentice Hall Hispanoamericana.

Roos, K. A. & Write C. R. B. (1990): Matemáticas Discretas, Segunda Edición, 1990 de la Primera en Español, Prentice Hall Hispanoamericana

Silverman,B., G., Hexmoor, H., H., Rastogi, R. and Chang, J.,(1988): Distributed Expert Systems:FACILITY ADVISOR, Expert System Aplications to Telecommunications, Ed. Jay Liebowitz, Wiley Series in Telecommunications.

Silverman, B., G., Wening, G., R., and Wu, T. (1989): COPEing With Ongoing Knowledge Acquisition From Collaborating Hierarchies, SIGART Newsletter, April 1989, Number 108, Knowledge Acquisition Special Issue, pp 170-171.

Van Frassen G., M., (1971): Formal Semantics and Logic, Macmillan, New York.

Wolf, W., A., (1989): Knowledge Acquisition from Multiple Experts, SIGART Newsletter, April 1989, Number 108, Knowledge Acquisition Special, Issue, pp 138-140.

Una Metodología de Adquisición de Conocimiento para Sistemas Expertos de Dignóstico Médico

Enrique Hinostroza, Pedro Hepp, Jorge A. González, Julián Mardones

Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile.
Casilla 6177 (143) Santiago 22, Chile.
Fax: (56-2) 5524054
E-mail: phepp@dcc.puc.cl

Resumen

En este trabajo se presenta una metodología de adquisición de conocimiento para sistemas expertos sobre un dominio acotado, i. e. diagnóstico médico. La metodología tiene su origen en el modelo Pizarrón [21] y se basa en la evidencia empírica del proceso de diagnóstico diferencial que los médicos aplican, la cual es modelada como etapas del proceso de adquisición. Se propone una estructuración jerárquica del diagnóstico diferencial que el médico realiza y el análisis posterior de cada una de las etapas definidas.

Luego, se presenta un ejemplo real de aplicación de la metodología de adquisición, donde se concluye que los aportes son: abstracción en el proceso de resolución, independencia en cada etapa definida y flexibilidad en la etapa de formalización de cada uno.

1. Introducción

Los Sistemas Expertos (SE) han demostrado ser una alternativa efectiva para solucionar problemas en los que no existen algoritmos eficientes de solución pero son tratables utilizando conocimiento heurístico[1]. En el área médica éstos han sido de gran utilidad para resolver muchos problemas de diagnóstico, entre otros, se pueden citar los siguientes: MYCIN [25], CASNET [31], PUFF (referido en [22]), TEIRESIAS [4], MEDAS [1], ONCOCIN [26], INTERNIST [23], CADUCEUS [23], GLADYS [28], CASEY [16]. Cada uno de estos sistemas soluciona, bien o mal, un problema de diagnóstico en el dominio de medicina. El hecho de existir varias experiencias en el diagnóstico médico permite considerarlos como una clase de SE.

Esta clase nace de la existencia de un factor común, cual es: el experto asociado siempre es un médico. Es decir, se basan en un modelo de razonamiento del experto que es similar en todos los casos ya que lo provee la formación propia del estudiante de medicina y la práctica de la profesión. Este factor tiene relevancia en el proceso de construcción del sistema experto y no en la implementación del mecanismo de resolución del problema de diagnóstico médico, ya que algunos están basados en métodos estadísticos (MEDAS), otros en reglas (MYCIN,

[1] Conocimiento heurístico es aquél adquirido en general por la experiencia.

ONCOCIN, TEIRESIAS), algunos en modelos causales (CASNET), o redes jerárquicas (INTERNIST, CADUCEUS) [28].

El proceso de construcción de un SE involucra, necesariamente, la adquisición del conocimiento (AC) del experto al resolver un problema. En la actualidad, para realizar la AC, no existen metodologías probadas, que permitan obtener conocimiento confiable, útil y completo [11]. Desde un punto de vista empírico, la AC sigue siendo el "cuello de botella" en la construcción de Sistemas Expertos [13] ya que es la etapa que consume más tiempo. Los autores concuerdan en señalar que los enfoques actuales basados en un acercamiento generalista es decir, que abarcan todas las topologías de problemas, no han logrado subsanar dicha dificultad. Además, afirman que es tiempo de pensar en tipos de adquisiciones de conocimiento según la topología del problema [3] [20].

A lo largo del tiempo se distinguen dos tendencias en el proceso de AC: adquisición automática y manual. La adquisición automática, básicamente, utiliza dos métodos: inductivos y estadísticos. Ambos han tenido cierto resultado y en la actualidad se investiga en la combinación de ellos para obtener mejores resultados [18]. En este trabajo nos dedicaremos a la investigación de la adquisición manual, por lo que dejaremos los procesos automáticos para trabajos futuros.

Para realizar la adquisición manual se han propuesto varios metodologías que pretenden ser de índole generalista [0] [24] [32]. En general, estas metodologías se basan en las 5 etapas de la AC definidas por Buchanan [20]: Identificación, Conceptualización, Formalización, Implementación y Pruebas. El apoyo al proceso en las etapas de Conceptualización y Formalización ha sido enfocado con diversas metodologías. La gran mayoría de ellas enfrentan la adquisición de conocimiento usando técnicas como: entrevistas, protocolos, simulaciones, etc., y diferentes representaciones intermedias como marcos, redes semánticas, diagramas de transición de estados, redes epistémicas, etc. [10] [13].

Una tendencia que complementa los modelos anteriores es el enfoque antropológico de la AC [8]. Este enfoque se basa en el conocimiento metodológico y teórico que ciencias como la antropología y sociología donde se propone analizar este proceso desde la perspectiva de los mecanismos de razonamiento del experto. Es decir, descubrir la metodología general que el experto aplica para resolver un problema y de ella obtener un modelo estructurado de AC.

El problema de AC en el área médica se ha tratado, en general, analizando el proceso de diagnóstico desde la perspectiva del médico enfrentado a un paciente [23] [2], es decir, se trata de modelar el proceso diagnóstico que el médico realiza y a través de este modelo, estructurar la

adquisición. En este trabajo se entrega una metodología de adquisición de conocimiento para Sistemas Expertos en la clase de diagnóstico médico que basa en un modelo empírico del proceso de diagnóstico y contempla básicamente la estructuración del dominio del problema en subproblemas diferenciables, posibles de analizar en forma estructurada.

En la primera parte de este trabajo se presenta un análisis del proceso de diagnóstico médico visto desde una perspectiva general. La metodología de adquisición que se propone a continuación, se basa en la modelación del resultado de este análisis. Por último, se presenta un caso práctico en el que se validan las ideas propuestas.

2. Fundamentos

El término diagnóstico tiene su origen en Grecia, *diagnostikos*, y significa *capaz de discernir*. Este significado, aplicado al área médica, implica poder seleccionar o distinguir una enfermedad[2] de acuerdo a los síntomas y signos que presente el paciente[3].

El proceso de diagnóstico en medicina se realiza a partir del análisis ordenado y sistemático de los síntomas y signos patológicos en la búsqueda de hipótesis iniciales, para concluir en una síntesis que explique cada uno de los hechos detectados a través de tres procedimientos claramente diferenciables y de la aplicación generalmente secuencial; éstos son:

- Anamnesis: recolección de los hechos de significación médica (síntomas u otros) en la vida del paciente, a partir del relato subjetivo de ellos por parte del enfermo. En general, permite comenzar a formular hipótesis diagnósticas, estrechando el dominio de hechos a considerar por los procedimientos ulteriores de diagnóstico.

- Examen físico: recolección de los hechos de significación médica objetiva (signos), obtenidos directamente por el clínico, estén aquéllos relacionados a una hipótesis diagnóstica inicial, en cuyo caso adquieren carácter confirmatorio, o no. En este último caso adquieren carácter evocador de nuevas hipótesis,

- Exámenes complementarios: Recolección de evidencias experimentales objetivas (exámenes de laboratorio) o bien de signos físicos a través de recursos tecnológicos estándar,

---

[2] Entendiendo por enfermedad a aquella entidad nosológica básica, caracterizada por la conjunción de un grupo de síntomas y signos clínicos determinados, que responden a una etiología u origen común y que representan una alteración fisiológica (funcional) y anatómica (estructural) precisa

[3] Historia Universal de la Medicina. Lain Entralgo Editor. SALVAT (Tomo 7). 1975. pp. 211-229.

que permiten poner a prueba una hipótesis diagnóstica consolidada en las etapas anteriores o bien comprobar su validez.

Al realizar el diagnóstico, el especialista utiliza, en general, estos tres procedimientos, pero los objetivos de cada uno de ellos se mezclan en una estructura de diagnóstico global. Es decir, cada uno de estos procedimientos son herramientas del especialista que utiliza al diagnosticar.

A grandes rasgos el médico realiza dos procesos distintos durante la recolección de datos con el paciente, con los cuales, espera converger a un diagnóstico. El primero es el de focalizar su campo de atención, es decir, llegar a un conjunto mínimo de entidades que compartan una sintomatología y signología común. A este proceso se le denomina diagnóstico sindromático[4]. Un síndrome habitualmente no identifica la causa precisa de una enfermedad, pero reduce el número de posibilidades y, a menudo, sugiere algunos estudios clínicos y de laboratorio especiales [33]. Es necesario destacar que este proceso no ocurre, necesariamente, en uno solo de los procedimientos antes descritos; puede ser necesario utilizar todos ellos para lograr esta focalización. El segundo proceso implica determinar una única explicación causal (etiológica), estructural (anatómica) y funcional (fisiológica) de entre el conjunto mínimo de diagnósticos asociados al síndrome. Este tiene como objetivo encontrar la evidencia suficiente para llegar a establecer el diagnóstico final. Al igual que la etapa anterior, este objetivo puede ser satisfecho a través de más de un procedimiento del proceso de diagnóstico.

Esta descripción del **proceso de diagnóstico en medicina** no siempre se cumple del mismo modo, ya que el énfasis en los diversos procedimientos diagnósticos puede variar frente a determinadas circunstancias (diagnóstico por descarte, diagnóstico pediátrico, diagnóstico del paciente comatoso, etc), pero constituye una técnica general que se aplica en la mayoría de los casos y, en particular, al tratado en este trabajo.

Otro aspecto que no se consideró en la descripción del proceso anterior es la experiencia del médico, es decir, la capacidad de realizar un diagnóstico en base a un conjunto mínimo de síntomas sin necesidad de seguir el camino completo de preguntas que un novato realizaría. Esta capacidad de obviar una serie de preguntas del protocolo normal de diagnóstico es lo que se considera heurísticas. Estas, en general, no tienen una explicación teórica conocida que fundamente su utilización.

---

[4] Síndrome es el grupo de síntomas y signos asociados a determinada alteración de función y relacionados entre sí por medio de alguna peculiaridad anatómica, fisiológica o bioquímica.

De acuerdo con estas características del proceso de diagnóstico es posible dividirlo en las siguientes etapas:

- Estratégica: En esta etapa el especialista utiliza conocimiento que le permite establecer el árbol de posibilidades de diagnóstico, es decir, determina el espacio de búsqueda acotado por restricciones que deben satisfacerse posteriormente durante el recorrido del espacio.

- Táctica: Una vez establecido el conjunto probable de estados sindromáticos (objetivos diagnósticos intermedios) y etiológicos (objetivos diagnósticos finales), el especialista utiliza el conocimiento sobre las características particulares de cada uno de las posibles síndromes o enfermedades para alcanzar y corroborar los diferentes estados diagnósticos.

- Heurísticas: Esta etapa complementa y hace más eficiente ambas etapas anteriores, al permitir al experto ahorrarse pasos propios de la sistematización lógica general que subyace a dichos procesos, en virtud de la aplicación de su conocimiento heurístico derivado de la experiencia.

Cada una de estas etapas implica la existencia de **distintos conocimientos** (conocimiento general, conocimiento particular, conocimiento empírico). Cada uno de ellos debiera ser **adquirido en forma separada** (o al menos identificado y tratado en forma separada). A continuación se analizan cada uno de estos procesos de adquisición dentro del marco de desarrollo de un sistema experto.

3. Diagnóstico y Adquisición de Conocimiento

En el año 1975 [7] se comienza a utilizar el modelo Pizarrón para la construcción de SE. Más tarde, en 1986 Penny Nii [21] formaliza el modelo de Sistemas Pizarrón para la construcción de Sistemas Expertos (SE). Este modelo tiene su base en la modularización del dominio del problema, estructurándolo de tal forma que los subdominios definidos correspondan a etapas diferenciables en el proceso global de resolución. Así, cada una de las etapas es tratada como un SE independiente del resto. Todos los SE así definidos se comunican a través de una memoria común que es llamada Pizarrón [6] [14].

En medicina este modelo ha sido propuesto por [2], desde una perspectiva generalista, es decir, en ella se plantea a cada una de las especialidades médicas representadas a través de fuentes de conocimiento en un sistema Pizarron. Este enfoque totalitario no es el caso de este trabajo ya que, desde una perspectiva práctica, una proposición de tal magnitud involucra tiempo y recursos con los que, actualmente no se cuentan. Además, sería una tarea de tal complejidad que, seguramente, no es el momento de enfrentarla, dado lo inmaduro de esta

ciencia. Ejemplos más específicos se encuentran en los Sistemas Expertos PUFF y MYCIN [22], que fueron reconstruidos utilizando una armazón (shell) de Sistemas Pizarrón.

Este enfoque de construcción de un SE es posible adaptarlo a la etapa de Adquisición de Conocimiento incorporando una interpretación semántica al control y programación (scheduling) del sistema, además de una sintáctica [27]. Esta adaptación depende del dominio de problemas a resolver y en el caso del diagnóstico en el área médica, es posible encontrar referencias que proponen esquemas similares [2] [23]. Estos esquemas se basan en la modularidad y enfoque jerárquico que el modelo Pizarrón provee.

En esta sección se analiza cada una de las etapas de diagnóstico definidas y se estructuran en una etapa de adquisición de conocimiento.

3.1 Adquisición estratégica

En esta etapa de adquisición el objetivo es modelar la estrategia de resolución que el médico tiene para resolver el problema de diagnóstico. Esta modelación se representa en una estructura que permite obtener una visión abstracta del proceso que el médico sigue al realizar el diagnóstico.

La estructuración de la estrategia de resolución del problema se obtiene al separar éste en módulos autocontenidos que representen los objetivo de diagnóstico intermedio (etapa de diagnóstico sindromático) y finales (etapa de diagnóstico etiológico o causal).

Para formalizar el conocimiento, los objetivos de diagnóstico definidos se representan a través de un grafo[5] que modela la estrategia de resolución del problema. En el grafo, los nodos representan las etapas de resolución y los arcos reflejan las relaciones entre éstas.

Por ejemplo, supongamos que un sistema experto en construcción se utilizará para apoyar el diagnóstico diferencial del síntoma Vértigo y que se está en la etapa de adquisición estratégica. Para obtener los resultados deseados de esta etapa, se le consulta al médico:

¿ Describa a grandes rasgos, qué etapas sigue al realizar el diagnóstico diferencial de esta enfermedad ?

La respuesta ideal sería aquella que permita estructurar el espacio de búsqueda y acotar al máximo su tamaño, por ejemplo:

---

[5] Para no restarle generalidad, en este trabajo se plantea la obtención de un grafo. En la práctica, se obtuvo un árbol.

Primero determino si el paciente presenta las características del síntoma vértigo, una vez definido esto, trato de determinar el origen anatómico de la lesión, separando su eventual origen en dos grandes grupos, sistema periférico y sistema central. Esta decisión se debe a la conveniencia de delimitar el número de posibles etiologías al de un tamaño manejable y finito [19], teniendo presente, además, que la frecuencia de lesiones centrales es muchas veces inferior a la de lesiones periféricas.

Una vez determinado el origen, en el caso de lesiones periféricas, investigo las enfermedades más fáciles de diagnosticar, estas se agrupan en ... etc.

Esta respuesta (que se obtiene en varias sesiones de adquisición), permite comenzar a estructurar el grafo estratégico que, en este ejemplo, sería:

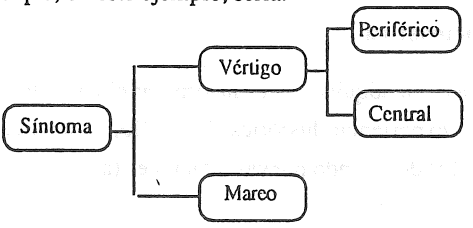


Figura 1. Resultado de una parte de la adquisición estratégica.

Este proceso sigue hasta completar el esquema general que el médico utiliza para realizar este diagnóstico y representarlo a través de un grafo.

3.2 Adquisición táctica

En este caso, para cada etapa de diagnóstico definida se establece un procedimiento para realizar el diagnóstico a ese nivel. La adquisición de cada una de estas etapas debe estar centrada en el objetivo que ésta cumple. Si se trata de una etapa intermedia, su objetivo será distinguir el camino a seguir para el siguiente nodo. En el caso de una etapa final, el objetivo es corroborar el camino de diagnóstico seguido hasta ahora.

Por ejemplo, siguiendo el caso anterior, se le hace la siguiente pregunta al médico:

¿Sin profundizar en otras etapas, cómo determina si el paciente presenta el síntoma vértigo?

Una respuesta que cumple los objetivos sería:

Si el paciente describe su molestia como una sensación de movimiento rotatorio de sí mismo o del entorno, o presenta síntomas acompañantes característicos, se puede concluir que sí corresponde al síntoma vértigo.

Otra pregunta, sería;

¿Supongamos que ya determinó que el paciente tiene el síntoma vértigo, cómo determina si la lesión es de origen central o periférico?

Como es de suponer, la respuesta a esta pregunta constituye otra etapa del sistema experto, que resuelve otra parte del diagnóstico diferencial. De esta forma, se avanza en la adquisición analizando cada etapa por separado, suponiendo que la información anterior es dada.

Para cada una de estas etapas pueden haber técnicas de adquisición y representaciones intermedias diferentes. Con esto se permite adecuar cada técnica y representación a la estructura de pensamiento del experto.

3.3 Adquisición heurística

El objetivo de esta etapa de la adquisición es incorporar el conocimiento inestructurado del experto, producto de su experiencia histórica. Para ello, se analiza y refina la estructura global del problema (grafo) definiendo nuevas relaciones (arcos) entre los objetivos de diagnóstico ya establecidos (nodos)[6].

Para reflejar estas nuevas relaciones en el comportamiento dinámico del sistema se modifica el contenido de cada etapa involucrada a través de un nuevo proceso de adquisición táctica.

La incorporación de heurísticas permite utilizar el conjunto mínimo de información que ayude a alcanzar el objetivo diagnóstico final más probable. En el caso de los objetivos diagnósticos intermedios se establecen relaciones más eficientes, y en el caso de los objetivos diagnósticos finales, se asocian diagnósticos alternativos relacionados (proceso de diagnóstico diferencial).

Siguiendo con el ejemplo anterior, supongamos que Enfermedad de Ménière y Fístula Perilinfática son diagnósticos diferenciales que en la práctica aparecen relacionados, pero están en grupos diagnósticos distintos; dada esta situación, una pregunta sería:

¿Suponga que en base a las preguntas que hemos hecho, se llega a pensar que el paciente sufre de Enfermedad de Ménière, pero no es totalmente seguro, cómo y en qué casos trataría de investigar si se trata de una Fístula Perilinfática ?

---

[6] En este proceso el árbol obtenido en la etapa de adquisición estratégica se transforma en un grafo.

La respuesta a esta pregunta obligaría a incluir un arco entre amabas enfermedades y, por lo tanto, a modificar el resultado de la adquisición táctica de estas etapas. Esto permite incorporar este conocimiento en forma estructurada, sin complicar las otras etapas.

A continuación se describe un caso práctico de la aplicación de esta técnica de adquisición. Este caso corresponde al ejemplo hasta ahora utilizado.

4. Diagnóstico Diferencial del Vértigo

El área médica es un excelente laboratorio de investigación para el campo de Inteligencia Artificial (IA), específicamente, Sistemas Expertos (SE) ya que en ella el diagnóstico es algo cotidiano. Por esto, este SE trata un problema de diagnóstico en medicina, es decir, a partir de ciertos síntomas y signos detectados en el paciente, llega a concluir un diagnóstico de la enfermedad. En este caso, el problema a resolver es el de diagnosticar una de las tantas patologías asociadas con el síntoma Vértigo[7].

El Vértigo es un síntoma frecuente en la práctica clínica, cuyo diagnóstico diferencial plantea muchas veces confusión o frustración al médico generalista [9] [15]. Por eso, no es raro que el diagnóstico de la patología causal sea sólo parcial o que se envíe a los pacientes en forma innecesariamente precoz al especialista. La mejoría de la eficacia y precisión en el diagnóstico diferencial por parte del médico generalista puede contribuir a un ahorro significativo de tiempo y recursos para el médico, el paciente y el sistema de salud general [30].

4.1 Adquisición estratégica

La realización de este proceso permitió estructurar un grafo que se muestra parcialmente en la figura 2.

---

[7] Vértigo se define como la sensación subjetiva o ilusión de movimiento, en la que la percepción más frecuente del paciente es que está girando en el espacio o bien que los objetos se mueven en el ambiente alrededor de él; sin embargo, a veces, la sensación es de desplazamiento en el espacio [9].

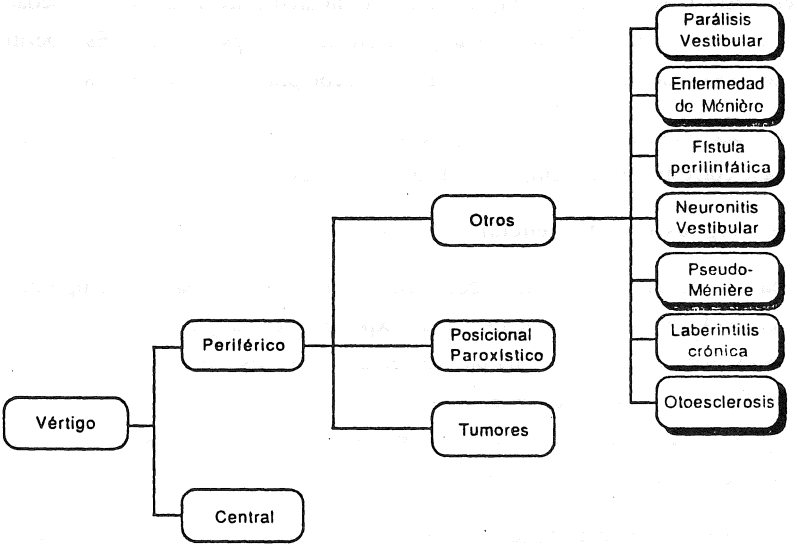


Figura 2. Parte del esquema jerárquico de diagnóstico del Vértigo.

Cada nodo de la figura 1 es un objetivo diagnóstico del problema del sistema experto en desarrollo y los arcos representan el camino de resolución a seguir. En el grafo se distinguen claramente objetivos diagnósticos intermedios (no todos explotados), a los que corresponden los siguientes nodos: Vértigo, Periférico, Central, etc. Por otra parte, los nodos: Parálisis Vestibular, Neuronitis Vestibular, Enfermedad de Ménière, etc., corresponden a objetivos de diagnóstico final.

4.2 Adquisición táctica

La adquisición en esta etapa, se basa en los objetivos de diagnóstico establecidos en la etapa anterior. Cada uno de ellos es analizado teniendo en mente las etapas siguientes.

Por ejemplo, en este caso, al hacer el análisis del objetivo diagnóstico intermedio denominado "Vértigo" (figura 1), se obtuvo el conocimiento necesario para construir un SE que discrimina Vértigos Centrales de Vértigos Periféricos.

La representación intermedia de cada una de los objetivos diagnósticos es variable y, en algunos casos, como en la figura 3, se estableció una secuencia de pasos que indica la transición de un nodo (Vértigo) a otro (Vértigo periférico)

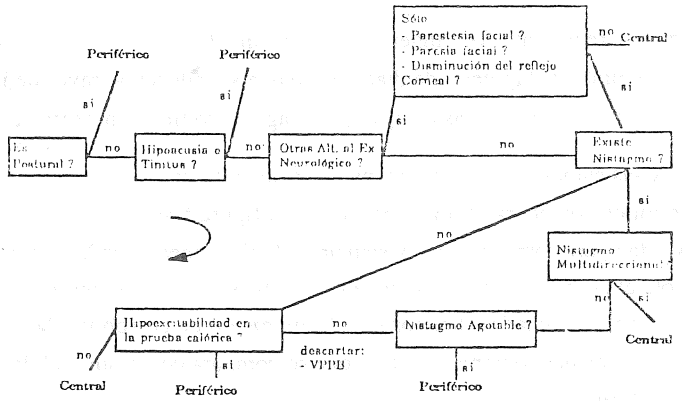


Figura 3. Diferenciación entre Vértigo Central y Periférico.

En otros casos, como en el objetivo diagnóstico denominado "Otros" en la Figura 2, se obtuvo una representación intermedia como la que se presenta en la figura 4.

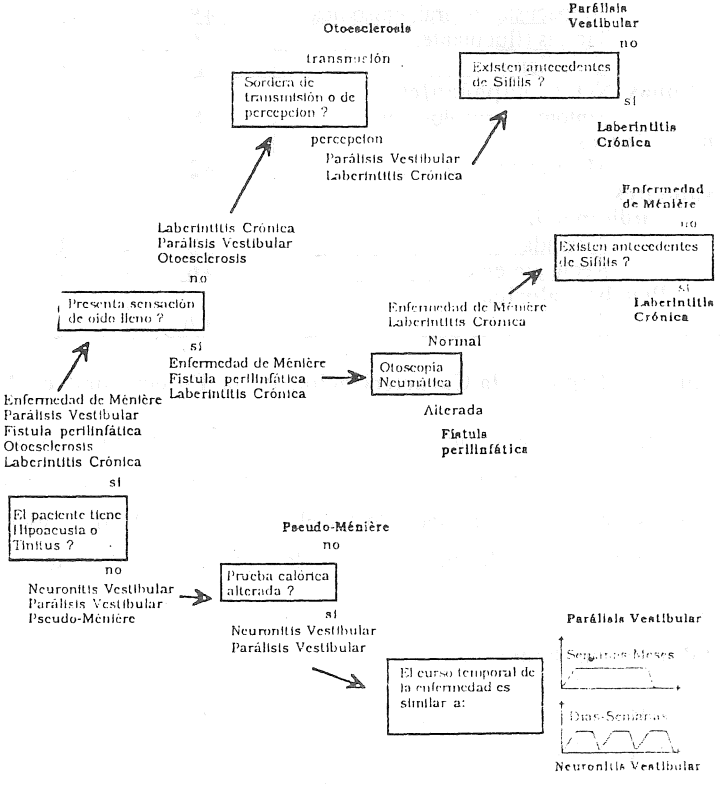


Figura 4. Esquema de decisión para el diagnóstico de "Otros Vértigos"

En la figura 4, la representación sugiere un mecanismo de razonamiento abductivo, ya que se parte de un conjunto de hipótesis de diagnóstico que se reducen a través de la evidencia empírica (antecedentes clínicos), hasta llegar a un diagnóstico final (en negrita en la figura).

En el caso de las etapas finales, como representación intermedia, se obtuvieron tablas de verdad que permiten corroborar el diagnóstico. En la figura 5 se muestra parte de una de estas tablas de verdad para corroborar la existencia de la Enfermedad de Ménière. En este caso, el experto anota el peso (importancia entre +10 y -10) asociado a la presencia o ausencia de un síntoma o signo en el cuadro diagnóstico. Al momento de corroborar la enfermedad, estos pesos son sumados para obtener un valor que se compara con el umbral de aceptación o rechazo de la enfermedad.

Síntoma o Signo	Presente	Ausente
Duración de la crisis:		
minutos a horas (20 min 4 hrs)	+9	-5
episódico (días a semanas)	+5	-6
Síntomas acompañantes:		
Hipoacusia (neural, episódica)	+9	-3
Tinitus (fluctuante,)	+6	-1
Oído lleno	1	-1
Síntomas NO acompañantes:		
Síntomas neurológicos	-8	+7
Antecedentes		
Ménière en la familia	+2	-1
Examenes		
Audiometría:		
Alterada	+8	-1
Reclutamiento	+6	-1
Prueba calórica		
Alterada	+6	-7

Figura 5. Parte de la Caracterización de la Enfermedad de Ménière

Estas tablas de verdad son criticadas en [29] debido a que en dominios muy extensos la recolección y procesamiento de todos los datos puede ser muy cara. En este caso se utilizan sobre dominios predefinidos y acotados lo que elimina esa complejidad. Además, el hecho de incorporar el peso de la evidencia ausente ha demostrado ser apropiado en el diagnóstico médico [23].

4.3 Adquisición heurística

Una vez caracterizadas todas las etapas del sistema, se obtiene un grafo global que modela el sistema (Figura 6). Este grafo relaciona todas las etapas predefinidas y además, incluye heurísticas que se agregan al completar el análisis global. Estas no fueron incorporadas desde

un comienzo en pos de un orden necesario en la adquisición de conocimiento de un dominio amplio. Las heurísticas se basan, principalmente, en razones de frecuencia de enfermedades y, a futuro, se pretende incorporar una base de datos que almacene la información de los diagnósticos para establecer una estadística propia, que permita definir nuevas heurísticas.

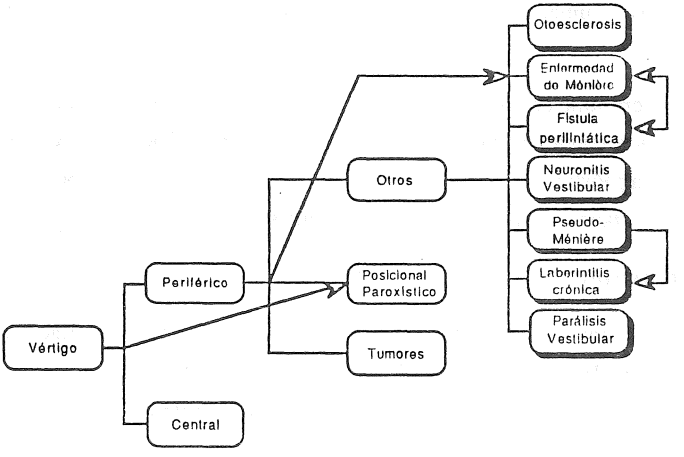


Figura 6. Estructuración de las etapas de diagnóstico del Vértigo.

Si al implementar el sistema experto se utiliza el modelo Pizarrón, este grafo puede ser utilizado para efectuar el meta-control del sistema [12].

5. Análisis de la Metodología de Adquisición de Conocimiento

El esquema de adquisición descrito tiene varias ventajas respecto de otros esquemas de adquisición tradicionales, éstas son:

- **Abstracción.** La utilización del modelo conceptual de los sistemas pizarrón, demanda un nivel de abstracción mayor al requerir de la definición de objetivos de diagnósticos diferenciables en la etapa de análisis general del problema. Esto repercute en la facilidad de análisis del problema completo, ya que se obtiene una visión global del proceso de adquisición a seguir.

- **Independencia.** Cada etapa definida es susceptible de ser analizada por un experto diferente, el cual sea especialista en ese dominio del problema. Esta característica permite la especialización del conocimiento y el máximo aprovechamiento del recurso humano (los expertos) en la construcción del sistema [2].

- **Versatilidad**. Al incorporar etapas de análisis independientes, es posible utilizar múltiples representaciones intermedias. Esto es posible, ya que, cada etapa definida se puede implementar utilizando una herramienta particular. Además, permite modelar en forma más exacta las distintas formas de razonar de los expertos[8].

Estos tres puntos resumen las ventajas de la utilización de este modelo en la adquisición de conocimiento.

Otro aspecto interesante de considerar en este método de adquisición es que debido a la independencia de cada etapa, los caminos del grafo resultante definen contextos separados para los diferentes diagnósticos finales (o intermedios) ya que es posible valorar los hechos o evidencias según la etapa o secuencia de etapas en que se consideren [23].

6. Conclusiones

Hasta la fecha no hay literatura de IA que sirva de guía útil de cómo hacer adquisición de conocimiento para Ingenieros del Conocimiento novicios [8]. El trabajo presentado es una guía para llevar a cabo la tarea de adquisición de conocimiento en el dominio de diagnóstico médico.

A través de este trabajo se ha mostrado la adaptación del modelo conceptual de los sistemas pizarrón en la etapa de adquisición de conocimiento de un sistema experto. Para ello se utilizó un sistema experto en diagnóstico diferencial del Vértigo que sirve como base empírica de esta adaptación. De la experiencia se concluye que este modelo aplicado a la adquisición provee de: abstracción en el proceso de resolución, independencia en cada etapa definida y versatilidad de la representación intermedia.

Estas características son de gran ayuda en la construcción de sistemas expertos de gran tamaño, por ejemplo, en sistemas expertos distribuidos [5] [17].

Como tarea futura queda la implementación de herramientas automáticas que apoyen la adquisición de conocimiento utilizando la metodología descrita.

---

[8] Se refiere a la característica híbrida de los mecanismos de razonamiento del ser humano, es decir, al resolver un problema, el experto utiliza múltiples esquemas de razonamiento (inducción, abducción, etc).

7. Agradecimientos

Este trabajo está respaldado por el Fondo Nacional de Ciencia y Tecnología de Chile, Proyecto #89-0591 y por el Programa de las Naciones Unidas para el Desarrollo, Proyecto CHI/ 88/022.

8. Bibliografía

0. Becker, L. Bartlett, R. Kinigadner, A. Roy, M. Using manufacturing Process Representation. *(AI EDAM) Artificial Inteligence for Engineering, Design, Analysis and Manufacturing*, 3 (1), 1989.

1. Ben-Bassat, M Carlson, W. C. Puri, V. K. Davenport, M. D. Schriver, J. A. Latif, M. Smith, R. Portigal, L. D. Lipnick, E. H. Weil, M. H. Pattern-based interactive diagnosis of multiple disorders: the MEDAS system. *IEEE Transactions on Pattern Analysis Machine Tnell,* 2:148-159, 1980.

2. Chandrasekaran, B. Towards a Taxonomy Of Problem Solving Types. *The AI Magazine*, Winter/Spring 1983.

3. Cullen, J. Bryam, A. The Knowledge Acquisiction Bottleneck: Time for Reassessment. *Expert Systems*, 5(3): 216 - 223, 1988.

4. Davis, R. Interactive transfer of expertise: acquisition of new inference rules *Artificial Intelligence* 12:121-157, 1979.

5. Durfee, E. H. Lesser, V. R. Corkill, D. D. Trends in Cooperative Distributed Problem Solving. *IEEE Transactions on Knowledge and Data Engineering* 1(1), 1989.

6. Engelmore, R. Morgan, T. Blackboard Systems. Addison-Wesley Publishing Company, Inc. 1988.

7. Erman, L. D. Lesser, V. R. A multi-level organization for problem solving using many diverse cooperating sources of knowledge *IJCAI 4*: 483-490, 1975.

8. Forsythe, D and Buchanan, B. Knowledge Acquisition for E.S: Some Pitfalls and Sugestion. *IEEE Transactions on Systems, Man and Cybernetics* 19(3), May/June 1989.

9. Goic, A. Chamorro, G. Mareos y Vértigos. En: Semiología Médica. Santiago, Publicaciones Técnicas Mediterráneo Ltda. 1987.

10. Greenwell, M. Knowledge Engineering for Expert Systems. Ellis Hordwood Books Series in Expert Systems, Chichester, 1988.

11. Hepp P.Introducción a la Adquisición de Conocimiento para Sistemos Expertos. *Apuntes de Ingeniería* (36):87-94, 1989.

12. Hinostroza, J. E. Hepp, P. Strategic Control in Blackboard Systems. *X International Conference of the Chilean Computer Science Society*, 23 al 27 de Julio, Santiago, Chile, 1990.

13. Hoffman R. The Problem of Extracting the Knowledge of Experts from the Perspective of Experimental Psychology, *AI Magazine* 8(2): 53-67, 1987.

14. Jagannathan, V. Rajendra Dodhiawala. Baum, L. S. Blackboard Architectures and Applications. Academic Press Inc. 1989.

15. Kerr, A. G. A symptomatic approach to vertigo. *J. Laryngol. Otol.* 97: 813, 1983.

16. Koton, P. A medical reasoning program that improves with experience. *Computer Methods and Programs in Biomedicine* 30:177-184, Elsevier, 1989.

17. Lesser, V. R. Corkill, D. D. The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks. En: Engelmore y Morgan,eds., Blackboard Systems, 1988. pp. 353-386.

18. Liang, T. Chandler, J. Han, I. Integrating Statistical and Inductive Learning Methods for Knowledge Acquisition. *Expert Systems with Applicationss* 1(4) 1990.

19. Martin, J. B. Approach to the patient with Nervous System Disease. En "Harrison's Principles of Internal Medicine. New York. McGraw Hill Book Co. 1987. 1 pp. 3.

20. Musen, M. Automated Generation of Model-Based Knowledge-Acquisition Tools. *Research Notes in Artificial Inteligence*, Pitman. London. Morgan Kaufmann Publishers, 1989. pp. 1-94.

21. Nii, H. P. Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures. AI Magazine 7 (2), 1986.

22. Nii, H.P. Aiello, N. AGE (Attempt to GEneralize): A Knowledge-Based Program for Building Knowledge-Based Programs. En: Engelmore y Morgan,eds., Blackboard Systems, 1988. pp. 251-280.

23. Pople, Jr., H. E. Heuristics Methods for Imposing Structure on Ill-Structured Problems: The Structuring of Medical Diagnostics. En P.Szolovits, eds., Artificial Inteligence in Medicine. Washinton, DC AAAS/Westview Press. 1988.

24. Rook, F. W. Croghan, J. W. The Knowledge Acquisition Activity Matrix: A System Engeneering Conceptual Framework. *IEEE Transactions on Systems, Man and Cybernetics* 19(3) May/June 1989.

25. Shortliffe, E. H. Computer-based Medical Consultations: MYCIN. New York Elsvier. 1976.

26. Shortliffe, E. H. Scott, A. C. Bishoff, M. B. Campbell, A. B. van Melle, W. Jacobs, C. D. ONCOCIN: an expert system for oncology protocol management. Proceedings of the 7th International Joint Conference on Artificial Intelligence, Vancouver, 1981. pp. 876-881.

27. Silverman, B. G. Chang, J. S. Kostas, F. Blackboard System Generator (BSG): An Alternative Distributed Problem Solving Paradigm. *IEEE Transactions on Systems, Man and Cybernetics* 19(2). March/April 1989.

28. Spiegelhalter, D. J. Kinll-Jones, R. P. "Statistical and Knowledge-based Approaches to Clinical Decision-support Systems, with an Application in Gastroenterology" J. R. Statist. Soc. A(1984), Part 1, pp. 35-77.

29. Szolovits, P. Pauker, S. G. Categorical and Probabilistic Reasoning in Medical Diagnosis. *Artificial Inteligence* 11:115-144, 1978.

30. Trappl, R. Porenta, G. Pharinger, B. Medical Expert System for Developing Countries: An application in primary Health care. *Micro-Electrinics Monitor*. (28) United Nations Development Programme. October 1989.

31. Weiss, S. M. Kulikoswki, C. A. Amarel, S. Safir, A. "A Model-Based Method for Computer-Aided Medical Decision-Making" Artificial Inteligence 11: 145-172. 1978.

32. Weitzel, J. R. Kerschberg, L. A System Development Methodology for Knowledge-Based Systems. *IEEE Transactions on Systems, Man and Cybernetics* 19(3), May/June, 1989.

33. Willson, J. D. Braunwald, E. Isselbacher, K. J. Petersdorf, R. G. Martin, J. B. Fauci, A. S. The Practice of Medicine. En: Harrison's Principles of Internal Medicine. New York. McGraw Hill Book Co. 1991.1 pp. 3.

Organización y documentación de programas grandes en 'C' (ansi)

Victor Manuel Toro C.
Grupo "Informatica Fundamental e Ingenieria de Software FIDIAS"
Departamento de Sistemas y Computacion - Universidad de los Andes
Apartado Aereo 4976
Bogota - COLOMBIA
(e-mail: vmtoro@andesco1.bitnet • Telex: 42343 UNAND CO • Fax: (+57) (1) 2841890)

RESUMEN

El desarrollo de programas grandes requiere establecer de antemano normas claras de particion, organizacion y documentacion. Esto es aún mas patente si el desarrollo se va a hacer en grupo. Así mismo, ceñirse a un conjunto claro de normas es requisito imprescindible para la añorada re-utilizacion de codigo. De otra parte, la posterior, costosa e inevitable labor de Mantenimiento puede ser grandemente beneficiada si a lo largo de toda la vida del programa se ha seguido algun estandar claro y conciso de organizacion y documentacion. Este articulo presenta una propuesta de estandar para organizacion y documentacion de programas grandes en C (Ansi) que ayuda a hacer mas agil la labor de desarrollo y mas certera la labor Mantenimiento.

CLASIFICACION ACM

D.1.4 Software/Programming Technique/Sequential Programming
D.2.3 Software/Software Engineering/Coding
D.2.7 Software/Software Engineering/Distribution and Maintenance

INTRODUCCION

> We must change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct the *computer* what to do, let us concentrate rather on explaining to *humans being* what we want the computer to do.
>
> Donald E. Knuth [KNUTH 84]

A lo largo de su corta historia, la Programacion ha sido vista como **Arte**, posteriormente como **Disciplina** y mas recientemente como **Ciencia**. El reto actual es que en los proximos años la programacion sea percibida como una verdadera **Ingeniería**.

En efecto hace unos 20 años se la consideraba un **Arte** ("The Art of Computer Programming: Fundamental Algorithms", [KNUTH 68]). Constaba de un conjunto de conocimientos aislados logrados por inspiracion o genialidad de sus inventores sin una teoria o lenguaje que los unificara. Solo era suceptible de ser aprendida por imitacion. Desarrollar un programa era un enorme esfuerzo de imaginacion, que eventualmnte convergia a algunas ideas utiles cuyo origen o correcion era muy dificil de justificar.

Años despues se calificaba a la Programacion como **Disciplina** ("The Discipline of Programming", [DIJESTRA 75]) se habian logrado identificar algunos patrones y esquemas unificadores dentro de los cuales se podia realizar una argumentacion ordenada y metodica. Se disponia de una serie de ejemplos y casos primorosamente resueltos, los cuales se ensenaban a los estudiantes por su gran valor formativo. Sin embargo en la practica rara vez se podian aplicar tan elegantes y rigurosos metodos, pues la incipiente teoria de programacion era excesivamente formal (i.e. llena de formulas) y compleja.

Desde comienzos de los 80 la Programacion se esta empezando a enfocar como **Ciencia** ("The Science of Programming", [GRIES 81]), es decir, sustentada por principios y teorias generales que permiten no solo explicar el porque de sus diferentes aspectos, sino ademas, y mas importante aun, que orientan clara, efectiva y eficientemente el proceso mismo de diseño [MEYER 88],[TORO 87, 91] y construccion de software [CARDOSO 90]. Inclusive, ya se vislumbra una etapa de madurez en la que preservando intacto el rigor se logra una razonable libertad e independencia del formalismo.

Para que en los proximos años la Programacion sea realmente una **Ingeniería** es necesario aumentar drasticamente la eficiencia y calidad del proceso de desarrollo de programas. Para lograr esto es requisito afinar y estandarizar metodos para las etapas anteriores a la Programacion (Analisis, Diseño, Especificacion), estandarizar la organizacion y la documentacion de programas, fijar criterios de calidad claros y verificables, y desarrollar herramientas que ayuden a aplicar dichas metodologias, y a ayudar/verificar el cumplimiento de los estandares de calidad, organizacion y documentacion [YUDKIN 88]. La juiciosa aplicación de dichos metodos, estandares y herramientas deberia facilitar la posterior, importante, costosa e inevitable labor de Mantenimiento [SCHWARTZ 82], [EINBU 88].

Este articulo presenta una propuesta de estandar para organización y documentacion de programas medianos y grandes en Ansi-C [K&R 88]. El estandar planteado da cabida a los avances en especificacion y correccion de software [GRIES 81], [CARDOSO 90], es compatible con cualquier metodo de Diseño (particularmente con el de las Jerarquias de Tipos Abstractos de Datos [TORO 87, 91]), y es susceptible de instrumentalizacion con herramientas concretas y simples. El articulo empieza entonces discutiendo el verdadero tiempo de vida de un programa. Posteriormente se resalta la necesidad de fijar y usar estandares de organizacion y documentación de programas. A continuacion se plantea la estrategia de uso del estandar propuesto. Finalmente se presenta el estandar de organizacion y documentacion. Dicha presentación se hace en tres etapas: inicialmente se describe el estandar para programas C en un solo archivo fuente, luego se presenta el estandar para programas C en varios archivos fuente, y finalmente se sugieren algunas normas generales (indentación, nomenclatura, ordenamiento).

EL TIEMPO DE VIDA DE UN PROGRAMA

> los programas realmente buenos viven para siempre y nunca terminaran de escribirse, al menos mientras dure el hardware e incluso mas
>
> Charles Simonyi (autor de Multiplan, Word, Excel) [LAMMERS 86]

En las primeras etapas de su vida estudiantil muchos programadores creen que los programas correctos y eficientes nunca necesitaran mantenimiento. Sin embargo, al menos tres razones importantes desvirtuan la anterior ilusion. Por un lado, las organizaciones en las cuales existen dichos programas y los usuarios en general, normal e inevitablemente evolucionan, generando nuevos procedimientos, produciendo nuevos datos, requiriendo nueva informacion. Por otro lado, los rapidos avances del hardware, pantallas, sistemas operacionales, manejadores de bases de datos, etc., periodicamente ponen en manos del programador facilidades y posibilidades que seria absurdo no aprovechar tanto en sus nuevos programas como en los ya existentes. Y finalmente, la busqueda de reutilizacion de software requeriere volver varias veces sobre el mismo codigo para ajustarlo o generalizarlo. El Mantenimiento de Software es entonces una importante, necesaria e inevitable labor.

En realidad y desafortunadamente, en muchos casos el idilico panorama que plantean los libros de Ingenieria de Software [PRESSMAN 87], no se puede llevar a cabo. Por diverso tipo de limitaciones, muchas veces no es posible adelantar concienzudamente las etapas de planeacion, analisis de requerimientos, especificacion formal de los requerimientos, diseño global, especificacion formal del diseño y programacion. No pocas veces solo se realiza superficialmente el analisis y el diseño y se entra en un ciclo programacion - correccion/ajustes - programacion - correccion/ajustes -.

La consecuencia es evidente: muchos departamentos de informatica terminan dedicando el grueso de sus recursos humanos a labores de mantenimiento y solo una fraccion a desarrollo de nuevas aplicaciones. Con el tiempo se llega a que en la mayoria de los sistemas grandes de software el costo y cantidad de trabajo acumulado en labores de Mantenimiento ha sobrepasado muy ampliamente el costo y cantidad de trabajo del Desarrollo original [BOEHM 84].

Obviamente la 'culpa' de esta situacion no radica unicamente en el no uso de estandares de programacion y documentacion. Sin duda la mayor parte del problema radica en que los metodos ofrecidos por la Ingenieria de Software son aun incapaces de enfrentar el caracter frecuentemente dinamico y volatil de los requerimientos y diseños.

De todas maneras, disponer y utilizar consciente y profesionalmente un estandar de organizacion y documentacion de programas contribuye significativamente a aclarar el panorama de programacion-mantenimiento de aplicaciones grandes.

IMPORTANCIA DE UN ESTANDAR DE DOCUMENTACION Y ORGANIZACION

> La profesora: a ver Montecristico: un señor compro una gallina en $1.721,35 y la vendio en $1.514,80 ¿ El señor gano o perdio en el negocio ?.
>
> Montecristico: El señor perdio en los pesos y gano en los centavos.
>
> Montecristo - Humorista Colombiano

La mejor documentacion para un programa es el programa mismo. Toda la demas documentacion (manuales de usuario, manuales de referencia, manuales de mantenimiento, documentos de diseño, etc.) desafortunadamente termina por desactualizarse. Ante cualquier labor de mantenimiento el documento que realmente se consulta es el listado mismo. La

demas documentacion usualmente solo es util para que el responsable de mantenimiento se familiarice con las caracteristicas generales del programa que debe mantener [EINBU 88].

En consecuencia, es de crucial importancia organizar los programas de una manera adecuada y proveerlos de comentarios e informacion estructural apropiados. Esta organizacion y la documentacion debe hacerse siguiendo esquemas y patrones claros y estandares. De lo contrario lo que se crearia seria aun mas variedad y desorden.

Sin embargo, organizar y documentar profesionalmente un programa es una labor que tambien requiere tiempo, experiencia y conocimiento, casi tanto como escribir buenos algoritmos !. La dificultad principal radica en que la documentacion de un programa se hace fundamentalmente en lenguaje natural (español, ingles, etc.). Y es un hecho comprobado que el lenguaje natural es tanto o mas dificil de usar que C ó Pascal, pues su sintaxis y semantica es obviamente menos rigurosa !.

El programador tendra entonces que dedicar su tiempo tanto a escribir las instrucciones que materialicen los algoritmos y estructuras de datos que tiene en mente, como a organizar y comentar adecuadamente su programa. Sin embargo, en la practica habitual casi siempre ocurre que la organizacion y documentacion de un programa solo se realiza en forma superficial y apresurada, despues de que el programa "funciona". Los programadores suelen creer que dedicar tiempo a documentar, a organizar, a escoger nombres apropiados para las variables, etc., es en buena medida desperdiciar su tiempo y retrasar la terminacion del programa. Inclusive es frecuente ver programadores que sienten remordimiento por el desperdicio de papel o los bytes extra que implica dejar algunas lineas en blanco para separar las diversas partes del programa. Muchos programadores solo se sienten "produciendo" cuando estan escribiendo instrucciones, y piensan que las labores de organizacion y documentacion son requisitos molestos y practicamente inutiles.

Nuestra experiencia directa, y particularmente la de muchos programadores realmente expertos [LAMMERS 86], es que es mucho mas eficiente organizar y documentar de acuerdo a algun estandar fijado de antemano, y que esta labor debe hacerse *al momento mismo en que se va escribiendo el programa*. Seria temerario afirmar que el tiempo total de desarrollo se acorta significativamente. Pero si podemos asegurar sin ninguna duda que se obtiene mayor calidad y que se sientan bases firmes para un mantenimiento mas agil y certero. El tiempo adicional necesario para la adecuada documentacion y organizacion o el papel o bytes extras que se requiere para mejorar la apariencia de un listado son los "centavos" que se mencionaban al comienzo de esta seccion. Los "pesos" son la calidad del producto y el tiempo de mantenimiento.

PRACTICA DEL ESTANDAR

> muchas cosas estaban prohibidas y las que no estaban prohibidas eran obligatorias ..."
>
> G. Garcia Marquez - "El Otoño del Patriarca"

Son varios los objetivos buscados al partir, organizar y documentar profesionalmente el codigo fuente:

- Descentralizar tanto como se pueda el desarrollo de cada uno de los modulos del programa, garantizando al mismo tiempo la coherencia global
- Facilitar la ubicacion de las diferentes secciones que componen un programa fuente grande, resaltando su estructura global
- Plasmar en el texto mismo del programa la informacion y las aserciones necesarias para facilitar la escritura de codigo correcto
- Proveer la informacion necesaria para que futuros programadores de mantenimiento puedan realizar su labor agil y certeramente

Hay que evitar extremos por defecto, no escribiendo comentarios suficientemente detallados, o por exceso escribiendo comentarios innecesarios o redundantes. La documentación debe ser una información precisa y breve, que se trasmite entre programadores altamente profesionales y competentes. La medida ideal podría ser entonces que el programador se planteara a sí mismo la siguiente pregunta: ¿ cuál es la mínima información necesaria para que yo mismo pudiera modificar este programa dentro de cinco años ?. ¡ Esa información mínima es la que hay que incluir en los comentarios !. Y ahora solo resta establecer una forma estándar de realizar las anteriores labores.

Es inutil intentar legitimar un estándar demostrando que cada una de las normas que lo componen es superior a cualquier otra norma posible [EINBU 88]. La estandarización es valiosa por sí misma. En la práctica es mejor acogerse a un solo estándar, posiblemente imperfecto, a debatirse entre varios candidatos a estándar (puntualmente mejores !), según el responsable de turno.

El estándar que aquí se plantea esta compuesto por una serie de recomendaciones generales, un esquema de organización de programas C en un archivo fuente, y un esquema de partición de programas C grandes en varios archivos fuente. Para facilitar la aplicación del estándar se han escrito una serie de plantillas, en cada una de las cuales se proveen todos los títulos y campos para comentarios, lo mismo que los `#include` necesarios para poder acoplar los diferentes archivos fuente. Existe entonces una plantilla por cada uno de los bloques o módulos en que se debe descomponer un programa C de acuerdo al estándar planteado.

Todas estas plantillas deberán ser colocadas en un directorio público, al cual tengan acceso todos los programadores. El proceso normal de programación de acuerdo al estándar consiste típicamente de:

- Empezar a editar un archivo inicialmente vacío.
- Leer desde el editor la plantilla correspondiente al contenido de dicho archivo (definición de constantes, declaración de prototipos de funciones, definición de variables globales, etc.)
- Diligenciar los campos que provee la plantilla hasta el nivel de detalle que el programador considere adecuado.
- Eliminar los campos que el programador no considere necesarios.
- A lo largo de la escritura del archivo fuente el programador podrá ir incorporando a su texto otras plantillas para documentar las nuevas funciones que vaya creando, hacer aserciones parciales o globales, dejar registro de modificaciones, etc.

LA FUNCION: UNIDAD BASICA DE UN PROGRAMA

Cada función esta precedida por un encabezado donde se presenta su descripción. La plantilla estándar corresponde a la necesidad maximal de documentación. Obviamente la mayoría de las funciones no requiere una documentación tan detallada como la que plantea dicha plantilla. Tal como se dijo anteriormente, hay que evitar extremos de sobre-documentación, o de documentación insuficiente. En consecuencia, el programador debera diligenciar cuidadosamente los campos que considere necesarios para documentar adecuadamente su función (recordar que el objetivo final es el mantenimiento !), y eliminar los campos innecesarios. En la página siguiente se presenta la plantilla "encfunc.i" (encabezado de función) y se describe el significado de cada uno de los campos.

```
/*****************************************************************************
* tipo nombre (tipo parm1, tipo parm2, tipo parm3,...)                       *
*****************************************************************************
* PROPOSITO:  ..................................................................*
* DESCRIPCION:  ................................................................*
*             ..................................................................*
* PARAMETROS:   tipo    nombre          comentario                           *
*               tipo    nombre          comentario                           *
*               tipo    nombre          comentario                           *
* RESULTADO:    tipo                    comentario                           *
* PRE:  ........................................................................*
* POST:  .......................................................................*
* OJO!:  .......................................................................*
*****************************************************************************
* AUTOR: i.i. apellido                                      FECHA: dd-mmm-91 *
*****************************************************************************/
tipo
nombre (tipo parm1, tipo parm2, tipo parm3,...)
{/*-------------------------------------------------------------------------*/
<definicion de variables locales>

<instrucciones>
/*-------------------------------------------------------------------------*/}
```

- PROPOSITO Que hace la funcion
- DESCRIPCION Breve informacion sobre como lo hace (solo cuando sea necesario)
- PARAMETROS Papel desempenado por cada uno de los parametros. Lo ideal es que la información suministrada en PROPOSITO y DESCRIPCION, junto con el sólo nombre del parametro ya provean la informacion suficiente. En consecuencia, el campo PARAMETROS debera ser diligenciado solo en los casos en que realmente se requiera.
- RESULTADO Tipo y significado del valor retornado por la funcion (solo cuando no sea claro a partir de la demas informacion)
- PRE Precondicion caracterizando las situaciones iniciales necesarias para el correcto desempeño de la funcion.
- POST Postcondicion caracterizando la situación final, luego del desempeño correcto de la función [GRIES 81, CARDOSO 90].
- OJO! Advertencia sobre posibles errores o limitaciones en la funcion.
- AUTOR y FECHA

Programas 'C' en un solo archivo fuente

La plantilla "unfuente.c" presente una organizacion de un programa C (ansi) en las siguientes secciones:

- #include s de los encabezados estandares.
- Declaración de constantes y macros.
- Declaracion de struct s, enum s, union s y typedef s (sin definir de variables)
- Declaracion de todos los prototipos de las funciones del programa (orden alfabetico).
- Definicion (e inicializacion) de las variables globales (sin declarar de nuevos tipos)
- Definicion de la funcion main
- Definicion de las demas funciones del programa.

```
/*****************************************************************************
* PROGRAMA: prg                                          FECHA: ---mes---- 1991 *
*****************************************************************************
* PROPOSITO                                                                  *
* DEL PROGRAMA: ..........................................................   *
*               ..........................................................   *
* SINTAXIS USO: prg [-opciones1] [-opciones2] file1 file2 ...                *
* OPCIONES-PARAMETROS.                                                       *
*      -x       ..........................................................   *
*      -y       ..........................................................   *
*      file1    ..........................................................   *
*      file2    ..........................................................   *
*****************************************************************************/
* AUTOR: i. i. apellido                                                      *
*****************************************************************************/

        /*-------------------------------------------------------+
        |          Encabezados de librerias estandares           |
        +-------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include <otros encabezados.h>

        /*-------------------------------------------------------+
        |    Declaracion de Constantes y Macros del programa     |
        +-------------------------------------------------------*/

#define .....   .....
#define .....    ....

        /*-------------------------------------------------------+
        | Declaracion de struct's, enum's, union's y typedef's   |
        +-------------------------------------------------------*/


... .... ......;

        /*-------------------------------------------------------+
        |        Prototipo de las funciones del programa         |
        +-------------------------------------------------------*/

tipo
nombre (  .  . .,    .  ...,  ... .  .);

tipo
nombre (... ....,  ... ....,  ... ....);

        /*-------------------------------------------------------+
        |     Definicion de variables globales del programa      |
        +-------------------------------------------------------*/

... ...,  ....,  ....,  ....;
... ....  ....,  ....,  ....;
... ...,  ....,   ...,  ....;
```

```
        /*-------------------------------------------------------+
        |                   Programa Principal                  |
        +-------------------------------------------------------*/

/*****************************************************************************
* int   main (int argc, char *argv[])                                        *
*****************************************************************************
* PROPOSITO:    ..............................................               *
* DESCRIPCION:  ..............................................               *
*               ..............................................               *
* PARAMETROS:   int     argc            # de parametros                      *
*               char    *argv[0]        nombre comando invocado            *
*               char    *argv[1]        primer parametro                   *
*               ....    ......          ........                           *
*               char    *argv[argc-1]   ultimo parametro                   *
* RESULTADO:    int     0               ejecucion exitosa                  *
*               int     -1              ejecucion fracasada                *
* PRE:          ..............................................               *
* POST          ..............................................               *
* OJO!          ..............................................               *
*****************************************************************************/
int
main (int argc, char *argv[])
{/*-------------------------------------------------------------------------*/
<definicion variables locales>

<instrucciones>
return 0;
/*-------------------------------------------------------------------------*/}

        /*-------------------------------------------------------+
        |    Definicion de las demas funciones del programa     |
        |                  (orden alfabetico)                   |
        +-------------------------------------------------------*/

/*****************************************************************************
* tipo nombre (tipo parm1, tipo parm2, tipo parm3,...)                       *
*****************************************************************************
* PROPOSITO:    ..............................................               *
* DESCRIPCION:  ..............................................               *
*               ..............................................               *
* PARAMETROS:   tipo    nombre          comentario                           *
*               tipo    nombre          comentario                           *
*               tipo    nombre          comentario                           *
* RESULTADO:    tipo                    comentario                           *
* PRE:          ..............................................               *
* POST.         ..............................................               *
* OJO!:         ..............................................               *
*****************************************************************************/
tipo
nombre (tipo parm1, tipo parm2, tipo parm3,..
{/*-------------------------------------------------------------------------*/
<definicion de variables locales>

<instrucciones>
/*-------------------------------------------------------------------------*/}

....
```

Programas 'C' en varios archivos fuente

Cuando un programa C se va a desarrollar en varios archivos fuente todos los archivos tendran en su nombre algún prefijo que identifique a que proyecto pertenecen. En la presentacion usaremos el nombre de proyecto "**pry**". Segun el estandar aqui propuesto un programa C en varios archivos fuente debe partirse y administrarse de la siguiente manera.

Archivos de Definiciones:

- Un archivo con el programa principal `main` y las funciones que dependen directamente del `main`. Se construye a partir de la plantilla "**pry_main.c**". El dueño de este archivo sera el jefe del grupo, y todos los demas miembros tendran unicamente permiso de lectura.
- Uno o mas archivos donde se definen grupos de funciones. El criterio de agrupacion puede ser por el tipo de tarea que desempeñan las funciones o por el objeto sobre el cual actuan, pero en cualquier caso, debe ser reflejo de la estructura global del diseño [TORO 91] (i.e., en un archivo irian las funciones de manejo de video, en otro las funciones de manejo de la base de datos, etc.). Cada uno de estos archivos se construye a partir de la plantilla "**pry_funx.c**". El dueño de cada uno de estos archivos sera programador que lo esta escribiendo y todos los demas miembros del grupo tendran unicamente permiso de lectura.
- Un archivo con la definicion de todas las variables globales. Se construye a partir de la plantilla "**pry_vglb.c**". El dueño de este archivo sera el jefe del grupo, y todos los demas miembros tendran unicamente permiso de lectura.

Archivos de Declaraciones:

- Un archivo con todas las declaraciones de constantes y macros globales del proyecto. A este archivo se le hace `#include` en todos los archivos de definiciones. Se construye a partir de la plantilla "**pry_defs.h**". El dueno de este archivo sera el jefe del grupo y todos los demas miembros tendran unicamente permiso de lectura.
- Un archivo con todas las declaraciones globales de `struct`s, `union`s, `enum`s y `typedef`s (sin definicion de variables). A este archivo se le hace `#include` en todos los archivos de definiciones. Se construye a partir de la plantilla "**pry_typs.h**". El dueño de este archivo sera el jefe del grupo, y todos los demas miembros tendran unicamente permiso de lectura.
- Un archivo de declaracion **extern** de todos los prototipo de funciones globales del proyecto. Asi, cada vez que algun programador del grupo pone a punto una nueva funcion global, debe actualizar este archivo de prototipos añadiendole una linea en la cual haga la declaracion `extern` del prototipo de la funcion que acaba de crear (y un breve comentario). A este archivo se le hace `#include` en todos los archivos de definiciones. Se construye a partir de la plantilla "**pry_prot.h**". El dueno de este archivo sera el jefe del grupo, y todos los demas miembros tendran permiso de lectura y escritura.
- Un archivo con la declaración **extern** de todas las variables globales (sin declaracion de nuevos tipos). A este archivo se le hace `#include` en todos los archivos de definiciones. Se construye a partir de la plantilla "**pry_vglb.h**". El dueño de este archivo sera el jefe del grupo, y todos los demas miembros tendran unicamente permiso de lectura.

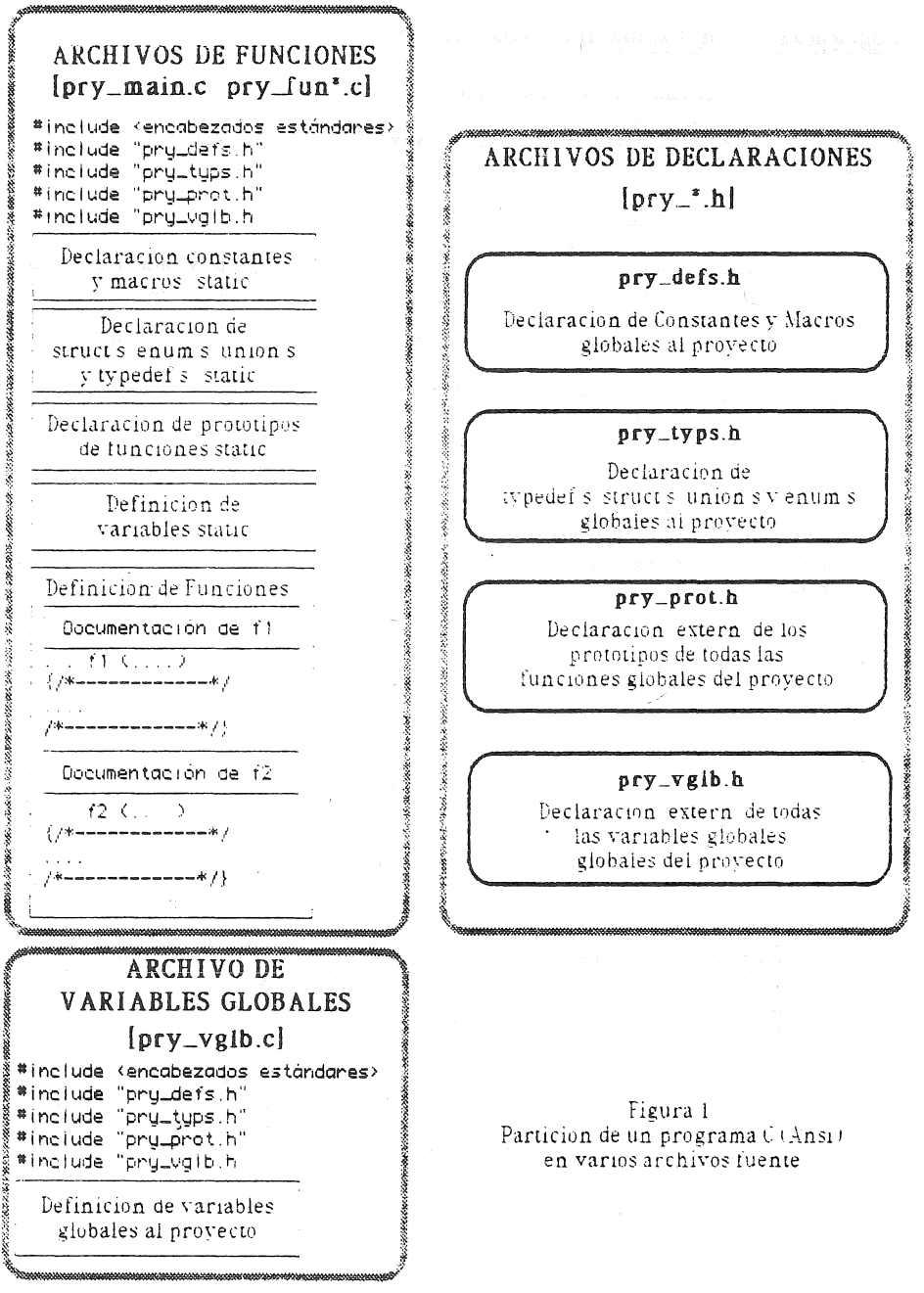


Figura 1
Particion de un programa C (Ansi) en varios archivos fuente

A continuación se muestran una a una las plantillas de base para construir cada uno de los archivos anteriormente mencionados

Plantilla **pry_defs.h**

```
/*************************************************************************
* PROYECTO: pry                                    FECHA: ---mes---- 1991 *
*************************************************************************/
/*       DECLARACION DE CONSTANTES Y MACROS GLOBALES AL PROYECTO         */
/*************************************************************************/

        /*-----------------------------------------------------+
        |                      CONSTANTES                      |
        +-----------------------------------------------------*/
#define ... ........
#define ..... ........

        /*-----------------------------------------------------+
        |                        MACROS                        |
        +-----------------------------------------------------*/
#define .....(...., ...., ) ........
#define .....(...., ...., ) ........
```

Plantilla **pry_typs.h**

```
/*************************************************************************
* PROYECTO: pry                                    FECHA: ---mes---- 1991 *
*************************************************************************/
/*    DECLARACION DE typedef's, struct's, union's y enum's GLOBALES      */
/*(En este archivo NO se deben definir variables ni declarar constantes) */
/*************************************************************************/

..... .... ..........;
.... ..... ..........,
```

Plantilla **pry_prot.h**

```
/*************************************************************************
* PROYECTO: pry                                    FECHA: ---mes---- 1991 *
*************************************************************************/
/*  DECLARACION DEL PROTOTIPO DE LAS FUNCIONES EXTERNAS DEL PROYECTO     */
/*           (En este archivo NO se deben definir funciones)             */
/*************************************************************************/

extern .... ......  (.... ....., .... ......, .... ......,  );
        /* proposito ..................................................*/

extern .... ......  (.... ....., .... ......, .... ......,  );
        /* proposito ..................................................*/
```

Plantilla **pry_vglb.h**

```
/*************************************************************************
* PROYECTO: pry                                    FECHA: ---mes---- 1991 *
*************************************************************************/
/*      DECLARACIÓN 'extern' DE LAS VARIABLES GLOBALES DEL PROYECTO      */
/*    (No definir variables ni declarar nuevas categorias de objetos)    */
/*************************************************************************/

extern .... ..........;     /* papel de la variable */
extern .... ..........;     /* papel de la variable */
```

```
/*****************************************************************************
* PROYECTO: pry          MODULO: pry_main.c      FECHA: ---mes---- 1991 *
*****************************************************************************
* PROPOSITO                                                                 *
* DEL PROYECTO: .... .. ..... ..... ..... ..... ..... ..... ..... ..... *
*               ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... *
* SINTAXIS USO: pry [-opciones1] [-opciones2] file1 file2 ...               *
* OPCIONES-PARAMETROS:                                                      *
*     -x        ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... *
*     -y        ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... *
*     file1     ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... *
*     file2     ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... *
*                                                                           *
*    PROPOSITO GLOBAL DE LAS FUNCIONES IMPLEMENTADAS EN ESTE MODULO         *
* ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... *
* ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... *
*****************************************************************************
* AUTOR DE ESTE MODULO: i. i apellido                                       *
*****************************************************************************
* AUTORES DEL   i.i. apellido      i.i. apellido      i.i. apellido         *
*  PROYECTO:    i. i  apellido     i.i. apellido      i. i. apellido        *
*****************************************************************************/

        /*-----------------------------------------------------------+
        |          Encabezados de librerias estandares              |
        +-----------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <otros encabezados.h>

        /*-----------------------------------------------------------+
        |    Declaracion de Constantes y Macros del proyecto        |
        +-----------------------------------------------------------*/
#include "pry_defs.h"

        /*-----------------------------------------------------------+
        | Declaracion de struct's, enum's, union's y typedef's      |
        +-----------------------------------------------------------*/
#include "pry_typs.h"

        /*-----------------------------------------------------------+
        |   Prototipo de las funciones externas del proyecto        |
        +-----------------------------------------------------------*/
#include "pry_prot.h"

        /*-----------------------------------------------------------+
        |    Declaracion de variables globales del proyecto         |
        +-----------------------------------------------------------*/
#include "pry_vglb.h"

        /*-----------------------------------------------------------+
        |          Declaracion de Constantes y Macros               |
        |           usados unicamente en este modulo                |
        +-----------------------------------------------------------*/
#define .... ....
#define .... ....
```

```
        /*-------------------------------------------------------+
        | typedef's, struct's, union's y enum's de este modulo  |
        +-------------------------------------------------------*/

   .. ........ .....;
   .   ......   ......;

        /*-------------------------------------------------------+
        |  Prototipo de las funciones 'static' de este modulo   |
        +-------------------------------------------------------*/
static ..... ...... (.... ....., .... ........, .... ........,   );
static ..... ...... (.... ....., .... ........, .... ........,   );

        /*-------------------------------------------------------+
        |    Definicion de variables 'static' de este modulo    |
        +-------------------------------------------------------*/
static ..... ......;
static ....  .......;

        /*-------------------------------------------------------+
        |                  Programa Principal                   |
        +-------------------------------------------------------*/
/*************************************************************************
* int   main (int argc, char *argv[])                                    *
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
* PROPOSITO:    ..................................................... *
* DESCRIPCION:  ..................................................... *
*               ..... ..... ..... ..... ..... ..... ..... ..... ..... *
* PARAMETROS:   int     argc            * de parametros              *
*               char    *argv[0]        nombre comando invocado      *
*               char    *argv[1]        primer parametro             *
*               char    ......          ........                     *
*               char    *argv[argc-1]   ultimo parametro             *
* RESULTADO:    int     0               ejecucion exitosa            *
*                       -1              ejecucion fracasada          *
*************************************************************************/
int main (int argc, char *argv[])
{/*-------------------------------------------------------------------*/
<definicion variables locales>

<instrucciones>
/*-------------------------------------------------------------------*/}

        /*-------------------------------------------------------+
        |   Definicion de las demas funciones de este modulo    |
        +-------------------------------------------------------*/
/*************************************************************************
* tipo nombre (tipo parm1, tipo parm2, tipo parm3,...)                   *
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
* PROPOSITO:    ..................................................... *
* DESCRIPCION:  ..................................................... *
*               ..... ..... ..... ..... ..... ..... ..... ..... ..... *
*************************************************************************/
tipo nombre (tipo parm1, tipo parm2, tipo parm3,...)
{/*-------------------------------------------------------------------*/
<definicion de variables locales>

<instrucciones>
/*-------------------------------------------------------------------*/}
```

Plantilla **pry_funx.c**

```
/*****************************************************************************
* PROYECTO: pry          MODULO: pry_funx.c      FECHA: ---mes---- 1991 *
*****************************************************************************
*    PROPOSITO GLOBAL DE LAS FUNCIONES IMPLEMENTADAS EN ESTE MODULO         *
* ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... *
* ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... *
*****************************************************************************
* AUTOR DE ESTE MODULO:                                                     *
*****************************************************************************
* AUTORES DEL    i.i. apellido      i.i. apellido      i.i. apellido        *
*  PROYECTO:     i.i. apellido      i.i. apellido      i.i. apellido        *
*****************************************************************************/

<*includes y declaraciones estáticas idénticos a "pry_main.c">
...
        /*-------------------------------------------------------+
        |      Definicion de las funciones de este modulo       |
        +-------------------------------------------------------*/

/*****************************************************************************
* tipo nombre (tipo parm1, tipo parm2, tipo parm3,...)                      *
*****************************************************************************
* PROPOSITO:     ..................................................... *
* DESCRIPCION:   ..................................................... *
*                ..................................................... *
* PARAMETROS:    tipo    nombre          comentario                     *
*                tipo    nombre          comentario                     *
* RESULTADO:     tipo                    comentario                     *
* PRE:           ..................................................... *
* POST           ..................................................... *
*****************************************************************************/
tipo
nombre (tipo parm1, tipo parm2, tipo parm3,...)
{/*-----------------------------------------------------------------------*/
<definicion de variables locales>

<instrucciones>
/*-----------------------------------------------------------------------*/}
....
```

Plantila: **pry_vglb.c**

```
/*****************************************************************************
* PROYECTO: pry          MODULO: pry_vglb.c      FECHA: ---mes---- 1991 *
*****************************************************************************
*           DEFINICION DE LAS VARIABLES GLOBALES DEL PROYECTO               *
* (NO declarar struct,s, union's, enum's ni typedef's en este archivo)      *
*****************************************************************************/

<*includes idénticos a "pry_main.c">
...
        /*-------------------------------------------------------+
        |   Definicion de las Variables Globales al proyecto    |
        +-------------------------------------------------------*/
....    ........, .....,  .......;
....    ........, .....,  .......;
```

Manejar 'a mano' tantos archivos fuente puede ser causa de errores de coherencia o de ineficiencias (encadenar versiones desactualizadas, recompilar innecesariamente todos los archivos...). Es entonces importante apoyarse en las herramientas usualmente disponibles para manejo de programas partidos en varios fuentes, como **make** [KERNIGHAN 84, cap. 8] o archivos **.prj** [BORLAND 88]. A continuacion se muestran las plantillas estandar para el uso de tales herramientas.

Plantilla **makefile**

```
#=============================================================================#
# makefile del proyecto "pry"                                  ---mes--- 1991 #
#*****************************************************************************#
# INSTRUCCIONES.                                                              #
#   - Cambie "pry" por el nombre abreviado de su proyecto.                    #
#   - Ajuste el nombre de los archivos pry_fun1.c ... pry_funN.c, y com-      #
#     plete los 'path' necesarios.                                            #
#   - Revise y eventualmente redefina las macros CC, CFLAGMOD, CFLAGEXE.      #
#=============================================================================#

# +-----------------------------------+
# | Re-Definicion de Macros estandares |
# +-----------------------------------+
# Coloque en la siguiente definicion su compilador C preferido
CC = "/usr/local/bin/gcc"

# Coloque en la siguiente definicion los flags
# deseados para la compilacion de cada modulo
CFLAGMOD = "-ansi"

# Coloque en la siguiente definicion los flags
# deseados para la compilacion del ejecutable principal
CFLAGEXE =

# +-----------------------+
# | Definicion de Macros |
# +-----------------------+
OBJETOS =       pry_vglb.o pry_main.o pry_fun1.o pry_fun2.o . . pry_funN.o
FUENTES =       pry_vglb.c pry_main.c pry_fun1.c pry_fun2.c ... pry_funN.c
ENCABEZADOS =   pry_defs.h pry_typs.h pry_prot.h pry_vglb.h

# +-----------------------------------+
# | Re-Definicion de Reglas implicitas |
# +-----------------------------------+
.c.o:
        $(CC) -c $(CFLAGMOD) $<

# +-------------------------------------------------+
# | Reglas para armar el programa pry y sus modulos |
# +-------------------------------------------------+

pry:        $(OBJETOS) ;   $(CC) $(CFLAGEXE) -o pry $(OBJETOS)
pry_vglb.o: $(ENCABEZADOS) pry_vglb.c
pry_main.o: $(ENCABEZADOS) pry_main.c
pry_fun1.o: $(ENCABEZADOS) pry_fun1.c
....
pry_funN.o: $(ENCABEZADOS) pry_funN.c
```

Plantilla **pry.prj**

```
# Archivo de definicion del proyecto pry para Turbo-C (2.0)
# Instrucciones:
#   - Cambie "pry" por el nombre abreviado de su proyecto.
#   - Ajuste el nombre de los archivos pry_fun1.c ... pry_funN.c, y com-
#     plete los 'path' necesarios.
#   - Hecho lo anterior, elimine esta linea y las anteriores !!!.

pry_vglb ( pry_defs.h pry_prot.h pry_typs.h pry_vglb.h )
pry_main ( pry_defs.h pry_prot.h pry_typs.h pry_vglb.h )
pry_fun1 ( pry_defs.h pry_prot.h pry_typs.h pry_vglb.h )
pry_fun2 ( pry_defs.h pry_prot.h pry_typs.h pry_vglb.h )
....
pry_funN ( pry_defs.h pry_prot.h pry_typs.h pry_vglb.h )
libreria_especial_1.lib
....
```

ALGUNAS RECOMENDACIONES GENERALES

> puedo decir si un programa es malo con un simple vistazo del listado. Obviamente, el solo listado no bastaría para afirmar que sea bueno, pero si desde tres metros el listado tiene mal aspecto, puedo garantizar que el programa no ha sido escrito con cuidado.
>
> Charles Simonyi (autor de Multipian, Word, Excel). [LAMMERS 86]

Son varias las publicaciones con recomendaciones generales sobre nomenclatura, indentación, sintaxis y estilo, orientadas a aumentar la legibilidad y facilitar el mantenimiento de programas C (GNU 90). [LEDGARD 87]. A continuación mencionaremos unas pocas reglas básicas.

No alterar la sintáxis de C

No cambiar la apariencia ni las estructuras de control de C a través del preprocesador [ARNOLD 85]. En efecto, la versatilidad del preprocesador de C permite extender o cambiar la sintaxis del texto a compilar, de manera que se puedan compilar programas escritos con sintaxis intermedias entre C y Pascal (u otros lenguajes). Son entonces nocivas macros del estilo:

```
#define    begin   {
#define    end     }
#define    if      if (
#define    then    )
#define    and     &&
#define    or      ||
#define    not     !
#define    equal   ==
```

pues generan grandes complicaciones para el mantenimiento y dificultan la portabilidad entre programadores (¡aún más importante que la portabilidad entre computadores!). Además, contribuye a impedir que el programador adopte realmente el estilo de C.

Nomenclatura

Los programas grandes deben crear y bautizar una gran cantidad de objetos (variables, procedimientos, constantes, archivos). Es entonces conveniente establecer criterios de nomenclatura, de manera tal que el nombre mismo del objeto provea alguna información sobre su categoría, características, ... [FOSTER 87]

- Los nombres de las variables importantes (i.e., excluyendo contadores locales, apuntadores temporales, etc.) deberán construirse a partir de un sustantivo eventualmente seguido de un adjetivo, abreviados. Ejemplos válidos son `linea_activa`, `promed_total`, `limite_credito`, `acum_ventas`, `colum_cursor`, `color_actual`, `ventas_dia_anterior`, ...
- Los nombres de las funciones deberán construirse a partir de un verbo en infinitivo, eventualmente seguido de un sustantivo y un adjetivo, abreviados. Ejemplos válidos son `verificar_nombre`, `acumular_venta`, `cargar_linea_nueva`, `guardar_plano`, ...
- Las variables apuntadoras deberán tener nombre de variable, seguido de un sufijo (**`_ptr`**) que resalte su condición (`prox_linea_ptr`, `sig_instruccion_ptr`, `ant_columna_ptr`, ...)
- Los nombres de categorías de objetos deberán estar construidos a partir de un sustantivo y estar seguidos de un sufijo que los distinga de los objetos en sí (**`_str`**: `struct`; **`_enm`**: `enum`; **`_uni`**: `union`; **`_typ`**: `typedef`)

```
struct cliente_str {
    ---- ------,
    ---- --------;
    ---  -----;
};
enum comandos_enm {borre, suba_cursor, baje_cursor, inserte, termine};
```

Declaraciones y Definiciones

- Evitar declarar categorías de objetos y definir objetos de dicha categoría en la misma frase. Preferiblemente declarar todas las categorías de objetos en la sección adecuada de la plantilla, y definir posteriormente las variables importantes, de preferencia en definiciones separadas. Así por ejemplo, en lugar de escribir

```
struct cliente_str {
    ---- ------;
    --   -------,
    ---  -----;
} cliente_viejo, cliente_nuevo, cliente_actual;
```

es más claro escribir las frases:

```
struct cliente_str {
    ---- ------,
    --   -------;
    ---  -----;
};

struct cliente_str  cliente_actual;
struct cliente_str  cliente_nuevo;
struct cliente_str  cliente_viejo;
```

Indentación

Un aspecto un tanto marginal, pero que suele dar lugar a acaloradas discusiones, es el esquema de indentacion. Tal como lo señalan los autores de C [K&R 88, pag. 10], lo realmente importante es que todos los miembros del equipo de desarrollo usen consistentemente algun estilo. De hecho es facil construir herramientas que reformateen un programa de acuerdo a diversos esquemas de indentacion [ECUYER 87] [CAMERON 88].

El esquema aqui presentado es una ligera variante del estilo presentado por los autores de C [K&R 88]. Nuestro esquema resalta el alcance del control, de manera tal que de un solo vistazo, y sin necesidad de ningun analisis sintactico, se ubiquen fácilmente los limites de control de una instruccion. El uso consistente de este estilo nos ha dado excelentes resultados, permitiendo inclusive con el solo reformateo de antiguos programas, encontrar graves errores que habian permanecido ocultos largo tiempo [K&R 88, pag. 56].

Las reglas del esquema propuesto son las siguientes: a nivel mas externo las instrucciones empiezan en la columna 1, las estructuras de control se cierran siempre y explicitamente a la misma altura que se abren; los cuerpos de instrucciones de control se indentan 4 columnas; los fines o comienzos de seccion dentro de una misma estructura se indentan 2 columnas. El esquema de indentacion tiene entonces la siguiente apariencia.

```
while (expresion)
    frase simple
;

for (----; ----; ----)
    frase simple
;

do {
    frase simple;
} while (expresion);

if (expresion)
    frase simple
;

if (expresion)
    frase simple;
  else
    frase simple
;

if (expresion) {
    --------;
    --------;
  }
  else
    frase simple
;
```

```
while (expresion) {
    --------;
    --------;
}

for (----; ----; ----) {
    --------;
    --------;
}

do {
    -------;
    -------;
} while (expresion);

if (expresion) {
    -------;
    -------;
};

if (expresion) {
    --------;
    --------;
  }
  else {
    --------;
    --------;
  }
;

if (expresion)
    frase simple;
  else {
    --------;
    --------;
  }
;
```

Finalmente, se presentan los esquemas de indentación del switch y de la sucesión de else-if's. Es de resaltar que la sucesión de else-if's, que implementa un esquema de control más general que el switch, se trata como una estructura en sí, y no como anidamiento de if's.

```
switch (expresion entera) {
  case <constante1>:
    ---------,
    break;
  case <constante2>:
    ---------;
    break;
  case <constante3>:
    ---------;
    ---------;
    break,
  default:
    ---------;
    break;
}
```

```
if (expresion1) {
    ---------;
    ---------,
  }
  else if (expresion2)
    ---------;
  else if (expresion3) {
    ---------;
    ---------;
  }
  else {
    ---------;
    ---------;
  }
;
```

Tal como se observa, ocasionalmente aparecen algunos pocos ; o } redundantes. Tales frases nulas son perfectamente validos en la gramatica de C y evidencian clara y brevemente el alcance del control en cada función, lo cual contribuye grandemente a la legibilidad del programa. Como ejemplo de comparación de presenta el siguiente trozo de código tomado de [K&R 88, pag. 134]:

```
int main (void)   /* Indentación de [K&R 88, pág. 134] */
{   int n;
    char word[MAXWORD];

    while (getword(word, MAXWORD) != EOF)
        if (isalpha(word[0]))
            if ((n = binsearch(word, keytab, NKEYS)) >= 0)
                keytab[n].count++;
    for (n = 0; n < NKEYS; n++)
        if (keytab[n].count > 0)
            printf("%4d %s\n", keytab[n].count, keytab[n].word);
    return 0;
}
```

```
int main (void)   /* Indentación con alcance de control explícito */
{/*-------------------------------------------------------------------*/
int   n;
char  word[MAXWORD];

while (getword(word, MAXWORD) != EOF)
    if (isalpha(word[0]))
        if ((n = binsearch(word, keytab, NKEYS)) >= 0)
            keytab[n].count++
        ;
    ;
;
for (n = 0; n < NKEYS; n++)
    if (keytab[n].count > 0)
        printf("%4d %s\n", keytab[n].count, keytab[n].word)
    ;
;
return 0;
/*-------------------------------------------------------------------*/}
```

CONCLUSIONES

> hay que estar siempre dispuesto a leer codigo escrito por otros y a dar nuestro propio codigo a revisión por otras personas. Hay que desear meterse en ese increible ciclo de realimentacion en el que se consigue que otras personas le digan a uno lo que está mal y uno lo acepta : No se puede permitir que pequeñas idiosincracias se interpongan y estorben esta realimentación ...".
>
> no creemos ni toleramos el sistema de la *primma donna* en el que solo porque alguien es muy bueno, se le permite que no comente su codigo, que no se comunique con los demas que arbitrariamente imponga sus creencias a los demas ."
>
> el equipo de programación ha de estar formado por personas que confien en si mismas y en los demas miembros del equipo, y que se respeten mutuamente puesto que se trata de un trabajo realmente intimo. Es como representar juntos en una misma obra musical"
>
> Bill Gates (fundador-director de MicroSoft) .LAMMERS 86.

La verdadera justificacion de una propuesta de estandar de organizacion y documentacion solo puede surgir de la practica. Durante casi tres años hemos venido usando y afinando el estandar aqui presentado en nuestros proyectos de desarrollo en numerosas tesis y proyectos de estudiantes de pregrado y postgrado y en asesorias directas a empresas. Podemos afirmar que una vez superada la resistencia inicial a alterar los habitos que traen los programadores habitos formados mas por costumbre que por reflexion los resultados son muy positivos No solo ha sido evidente que el desarrollo y mantenimiento se han vuelto mas coordinados y certeros sino tambien, que el uso de herramientas claves para el desarrollo de sistemas complejos (make, sccs, rcs) [KERNIGHAN 84,GNU 90] ha sido mas claro y ortogonal Inclusive en varias ocasiones hemos visto como los mas excepticos y reacios terminaron convirtiendose en los mas entusiastas defensores del estandar convencidos de que la creatividad e individualidad tiene mejor campo de expresión en el analisis y el diseno que en detalles de indentacion nomenclatura forma de documentar etc

> ¡Las plantillas aqui presentadas (y algunas otras) puede ser obtenidas en versión magnetica via e-mail, escribiendo a vmtoro@andescol.bitnet]

BIBLIOGRAFIA

[ARNOLD 88] Ken Arnold • "C Advisor: fun with the preprocessor": Unix Review pags. 73-77 april 1988

[BOEHM 84] Barry Boehm • "Software Economics", IEEE Transactions on Software Engineering

[BORLAND 88] Borland International • "Turbo C Version 2.0" © 1988

[CAMERON 88] Robert D. Cameron • "An Abstract Pretty Printer " IEEE Software. Nov 1988

[CARDOSO 89] Rodrigo Cardoso • Verificacion y Desarrollo de Programas". Publicaciones del Departamento de Sistemas - Universidad de los Andes

[DIJKSTRA 76] Edsger W. Dijkstra • "A Discipline of Programming" Prentice-Hall c 1976

[ECUYER 87] Pierre L Ecuyer • Formal Formatting Rules for Pascal Programs" The Journal of Systems and Software. Vol 7, pags 311-322. 1987

[EINBU 88] John M. Einbu • "An Architectural Approach to Improved Program Maintainability" Software Practice and Experience Vol 18, Nº 1 enero 88

[FOSTER 87] John R. Foster; Malcolm Munro • "A Documentation Method based on Cross-Referencing". IEEE Conference on Software Maintenance (Austin-Texas) 1987. pags. 181-185

[GNU 90] "GNU's is not Unix" project • Archivo /pub/gnu/standards.text (Noviembre2,1990); Obtenido via ftp anonimo al computador prep.ai.mit.edu (internet) en MIT

[GRIES 81] David Gries • "The Science of Programming". Springer-Verlag © 1981

[K&R 88] Brian W. Kernighan; Dennis M. Ritchie • "The C Programming Language (2º edition)". Prentice Hall © 1988

[KERNIGHAN 84] Brian W. Kernighan; Rob Pike • "The UNIX Programming Environment" Prentice Hall © 1984

[KNUTH 68] Donald E. Knuth • "The Art of Computer Programming: Fundamental Algorithms" (Vol. I). Addison-Wesley © 1968

[KNUTH 84] Donald E. Knuth • "Literate Programming". The Computer Journal (Wiley-Heyden). Vol. 27. Nº 2. pags. 97-111. 1984

[LAMMERS 86] Susan Lammers • "Programmers at Work: interviews" ("Programadores en Accion: entrevistas") MicroSoft Press © 1986 (Ediciones Anaya Multimedia S.A. © 1988)

[LEDGARD 87] Henry Ledgard; John Tauer • "Professional Software (Vol. II) Programming Practice". Addison-Wesley © 1987

[MEYER 88] Bertrand Meyer • "Object-Oriented Design". Prentice Hall © 1988

[PRESSMAN 87] Roger S. Pressman • "Software Engineering: a practitioner's approach (2ª edition)". McGraw-Hill © 1987

[SCHWARTZ 82] Barbara Schwartz • "Eight Myths about Software Maintenance". Datamation. pags. 125-128. august 1982

[TORO 87] Victor M. Toro C. • "Diseno Especificacion y Prototipificacion de Sistemas Interactivos". XIII Conferencia Latinoamericana de Informatica. Bogota - Noviembre 1987). Memo de Investigacion Nº 28. Facultad de Ingenieria. Universidad de los Andes

[TORO 91] Victor M. Toro C. • "Sistemas Interactivos = Tipos Abstractos de Datos + Control + Interfaz". Método y Estandar de Diseno de Sistemas Interactivos". Memo de Investigacion (proxima publicacion). Facultad de Ingenieria - Universidad de los Andes

[YUDKIN 88] Howard L. Yudkin • "Emerging trends present opportunities, challenges for standards developpment". IEEE Computer. pags. 57-58. August 1988

UN SISTEMA EXPERTO PARA EL DISEÑO DEL MODELO CONCEPTUAL DE UNA BASE DE DATOS

Jorge L. del Mar Arzola
Lima - Peru

RESUMEN

El diseño de una Base de Datos (BD) ha sido considerado un arte, en vez de una ciencia, debido a los problemas existentes en esta área de la Informática.

La esencia principal de estos problemas se deben a la falta de una adecuada estructuración en las metodologías de diseño de BD y a la carencia de soporte automatizado para el desarrollo de este proceso.

En la actualidad, un enfoque para solucionar estos problemas es utilizar las herramientas que nos brinda la Inteligencia Artificial (IA) para facilitar el proceso de diseño de una BD.

En el presente trabajo se ofrece una descripción de las características y capacidades de un Sistema Experto (SE) que brinda soporte en el diseño del modelo conceptual de una BD.

Además, se presentan los criterios de diseño que se utilizaron en el desarrollo del SE.

1.0 INTRODUCCION

El desarrollo actual de la tecnología informática pone en evidencia una tendencia muy general en las Organizaciones por diseñar e implementar una BD.

Tradicionalmente, el diseño de una BD se ha realizado en base a los criterios y la experiencia de un "experto diseñador de BD" que obtiene los requerimientos informáticos de la Organización mediante entrevistas a los usuarios, revisando los documentos y los Sistemas de Información (SI) existentes. Luego, el diseñador de la BD integra los requerimientos de información, de forma tal, que se integran los requerimientos de los SI de la Organización (existentes en el presente y los que se se desarrollen en el futuro).

A continuación, el usuario y el diseñador validan el modelo conceptual obtenido después de varias interacciones.

Cuando se obtiene una version final del modelo conceptual el diseñador transforma el modelo conceptual de la BD en un modelo lógico de datos requerido para su implementación en una forma compatible con el Sistema de Gestión de Base de Datos (SGBD) que se utilizará en la Organización (por ejemplo, el modelo relacional).

Finalmente, el modelo lógico de datos es transformado en una forma compatible con la tecnología de Software y Hardware específico de la Organización.

La utilización del enfoque tradicional presenta los siguientes problemas:

1. El diseño de una BD se convierte en una tarea difícil y compleja que requiere la utilización de un recurso escaso: El experto diseñador de BD. Por otro lado, la calidad del diseño de la BD depende mucho de la experiencia y el conocimiento del experto diseñador de BD.

2. En la actualidad, el diseño de BD es un arte que tiene poco soporte de apoyo automatizado lo cual hace que su éxito dependa de la intuición y la experiencia del experto diseñador de BD. El diseño de una BD no es una tarea trivial que pueda realizarse fácilmente utilizando sólo lápiz y papel.

3. El éxito en el modelamiento de una BD, depende de la habilidad para percibir la configuración organizacional e identificar las entidades y relaciones en la Organización, este proceso generalmente es realizado con una mínima participación de los usuarios, y después que el experto diseñador de BD termina de elaborar el modelo conceptual de la BD, presenta los resultados a los usuarios para su refinamiento (generalmente el período de tiempo que transcurre entre la especificación de requerimientos de información y la presentación de resultados tangibles es muy

dilatado).

4. Generalmente, el experto diseñador de una BD no esta familiarizado en un nivel de dominio de los sistemas aplicativos de la Organizacin.

Muchos investigadores en el área de BD han propuesto que los SE son una poderosa solución que brinda soporte al diseño de una BD de forma que se superen las deficiencias del diseño basado en el enfoque tradicional.

Los SE son programas basados en técnicas de IA que permiten aplicar aspectos del conocimiento humano, y ejecutar tareas que generalmente son realizados en base a la experiencia de los seres humanos.

La realización de un SE para el diseño de una BD se sustenta tomando como base las siguientes ventajas:

1. Si la metodología de trabajo de un experto diseñador de BD puede codificarse en un programa, entonces podría ejecutarse varias veces y aplicarse cuando se necesite, de esa manera, se superaría el problema que origina la escasez de expertos humanos.
2. Al codificar la metodología de un experto diseñador de BD en un SE que interactue con el usuario final, entonces el usuario - que es el que mejor comprende la aplicación - puede convertirse en el diseñador de su propia BD.
3. Se reduciría las etapas de diseño, desarrollo e implementación de una BD.
4. Las Organizaciones son entidades dinámicas que

pueden sufrir cambios rápidamente, lo cual trae como consecuencia que los requerimientos de información pueden variar, con un SE dichos ajustes se realizarían en forma oportuna e implementando rápidamente dichos requerimientos, aumentando la productividad y eficiencia al mejorar las condiciones de trabajo como consecuencia de la automatización del proceso de diseño de BD.

5. Algunas de las metodologías de diseño de BD tienen la orientación hacia el desarrollo de una aplicación y no desarrollan una BD integrada para la Organización, de forma que se brinde apoyo a todos los SI.

Por otro lado, la creciente necesidad de aumentar la eficiencia y efectividad en las Organizaciones ha generado, a su vez, un interés por analizar y perfeccionar los aspectos que inciden en el desarrollo de una BD, es por esta razón que es necesario desarrollar Software que brinde apoyo en el diseño de una BD de forma tal que los problemas ocasionados por un enfoque tradicional sean superados para maximizar las ventajas y beneficios que se obtienen de la filosofía inherente de la tecnología de BD.

El principal objetivo del presente trabajo es es llenar el vacío que en la actualidad existe en el área de BD en nuestro país, desarrollando un SE que brinde apoyo en el diseño del modelo conceptual de una BD orientado a un amplio rango de Organizaciones.

2.0 Sistemas Expertos y Diseño de BD.

El tema de la automatización del proceso de diseño de una BD ha sido enfocado por varios investigadores. NAVATHE y GADGIL argumentan que los principales problemas en el diseño de una BD son ocasionados por la carencia de una metodología de diseño estructurado y la falta de ayudas automáticas para desarrollar BD complejas NAVATHE (8), para dar solución a estos problemas, desarrollaron una metodología para automatizar el proceso de integración de "vistas" y, toma como base el modelo de vistas N - S NAVATHE (7).

En 1,984, J. MYLOPOULOS plantea que un SE puede recolectar información de las características de una BD que pueda utilizarse para generar el diseño de una BD en forma automática o semi - automática MYLOPOULOS (6).

Por otro lado, en 1,985, M. BOUZEGHOUB y un grupo de estudiosos de la Universidad de París aplican la tecnología de SE para producir un esquema interno a partir de las especificaciones de un esquema conceptual BOUZEGHOUB (2).

En 1,985, HAWRYSZKIEWYCZ desarrolla un SE que brinda apoyo al diseñador de una BD en el desarrollo del modelo E - R utilizando un dialogo en lenguaje natural HAWRYSZKIEWYCZ (3).

Luego, en 1,986, KAWAGUCHI desarrollo el

"INTELLIGENT INTERVIEW SYSTEM", que es un SE que brinda facilidades para que los usuarios construyan su BD en sus áreas de especialización KAWAGUCHI (4).

En 1,988, VEDA STOREY desarrolló el "VIEW CREATION SYSTEM", que es un SE que recolecta los requerimientos informáticos de los usuarios y a partir de esa información genera "vistas" de la BD STOREY (9)

Por otro lado, en 1,989 TAUZOVICH desarrollo "MODELER", un SE que recibe información (entidades, relaciones y atributos) en un lenguaje que es una extensión de "LESK" (un lenguaje de adquisición de conocimientos basado en lógica de primer orden muy similar al inglés y luego produce un modelo conceptua expresado en base al modelo E - R extendido TAUZOVICH (11).

En la actualidad, muchos investigadores están desarrollando estudios para desarrollar SE que automatizen el proceso de diseño de BD, el objetivo final es tener un SE que realmente funcionen como expertos diseñadores del BD y que produzcan diseños de alta calidad, el camino para alcanzar este objetivo es aún muy largo, pero es un reto interesante.

3.0 DESCRIPCION DE LAS CARACTERISTICAS FUNCIONALES DEL SE PARA EL DISEÑO DEL MODELO CONCEPTUAL DE UNA BD

3.1 DESCRIPCION DEL SE

El principal objetivo en el diseño del SE fue formalizar el proceso de diseño del modelo conceptual de una BD de forma tal que se puede automatizar dicho proceso y además se incremente la calidad y consistencia del diseño de la BD. Además, se debe reducir la dependencia de un recurso escaso y costoso: La persona experta en el diseño de una BD.

La meta específica en el diseño del SE fue formular un conjunto de reglas que sirvan como la base de conocimiento del SE y que permita la generación del modelo conceptual de la BD, es decir, formalizar el proceso de diseño del modelo conceptual.

El SE actúa como un diseñador de una BD en la determinación de los requerimientos informáticos de una Organización.

El SE esta diseñado para trabajar interactivamente, de forma tal que se consulte a los usuarios que utilizarán la BD sobre sus requerimientos informáticos.

Con la ayuda del usuario el SE emplea procedimientos de normalización para transformar los requerimientos de información en un conjunto de relaciones en Tercera Forma Normal que representa el modelo conceptual de la BD. La selección de la representación relacional fue arbitraria

porque de igual forma se pudo utilizar cualesquier otra representación (Red, E-R, etc.)

Un aspecto importante del SE es que es independiente de la Organización o de la aplicación en que se le utilize debido a que no contiene ningún conocimiento específico para una Organización o aplicación en particular.

Por otro lado, la base de conocimiento contiene la información sobre el proceso de diseño de una BD.

3.2 CARACTERISTICAS DEL SE

En la actualidad, el SE para el diseño del modelo conceptual de una BD esta compuesta por 1,000 líneas de código fuente escrito con TURBO PROLOG y se ejecuta en un microcomputador IMB PC/XT o compatible.

El TURBO PROLOG fue seleccionado como el lenguaje para el desarrollo del SE por la facilidad que brinda para la aplicación de la técnica de prototipo y la flexibilidad que provee al mantenimiento y el desarrollo posterior del SE.

La base de conocimiento del SE está compuesto por 50 reglas, muchas de las cuales a su vez contienen Sub-Reglas.

El SE ha sido diseñado especificamente para recolectar los requerimientos de información de los

usuarios de una Organización, enseñandole al usuario los principios del modelo E-R y brindando el soporte necesario para que el usuario pueda especificar el modelo conceptual de la BD.

Las características principales del SE son las siguientes:

a) Base de Conocimientos del SE

El conocimiento del SE esta basado en la teoría de diseño de BD y en nuestra experiencia en el diseño de BD. Además, la base de conocimiento ha aumentado en función de las experiencias que se ha tenido en la utilización y refinamiento del SE. El conocimiento del SE esta representado como un conjunto de procedimientos y de reglas de producción.

b) Rendimiento del SE

El SE ha sido probado utilizando varios problemas de diseño para una variedad de Organizaciones.

El SE trabaja muy bien con usuarios que tienen algún conocimiento previo de la teoría de BD. Para el caso de usuarios que no han tenido ninguna experiencia previa con los conceptos de BD es recomendable que utilicen el Tutor de Conceptos de BD (incorporado en el SE), antes de realizar el diseño de una BD, de forma tal que se puede explotar en su totalidad al SE.

El software desarrollado para dar soporte al diseño del modelo conceptual de una BD esta clasificado como un SE porque incorpora la experiencia humana en el diseño de BD.

c) Soporte del SE al Diseño de una BD.

El SE inicialmente se orienta a la fase de análisis de los requerimientos informáticos de una Organización. Luego, en el diseño conceptual se representan los requerimientos de información en término del modelo E-R. Finalmente, brinda un soporte al diseño del modelo lógico de datos utilizando el modelo de datos relacional.

d) Usuarios del SE

El SE ha sido diseñado para usuarios de una Organización de quienes no se espera que tengan conocimientos previos de diseño de una BD. Inicialmente, es preferible que los usuarios del SE tengan algún conocimiento de los conceptos básicos de BD. En todo caso, el SE brinda soporte para que el usuario aprenda los conceptos del diseño utilizando el modelo E-R y para que el usuario pueda aplicar dichos conceptos para resolver sus problemas.

La interfase con el usuario es de texto orientador para lo que se utiliza facilidades de "ventanas" y de "menús".

e) Independencia del Dominio de Aplicación

El SE es independiente del dominio de aplicación, debido a que no contiene ningún conocimiento previo sobre alguna aplicación en particular. Por otro lado, el SE tiene una capacidad limitada para aprender que atributos son comunmente utilizados en un dominio de aplicación particular.

3.4 Procedimientos del SE

La principal tarea en el diseño del SE fue la construcción de su Base de Conocimientos. La Base de conocimientos del SE representa la formalización del proceso de diseño del modelo conceptual de una BD.

La forma más simple de comprender la estructura interna del SE es revisar sus procedimientos. A continuación, describiremos los principales procedimientos del SE.

a) Procedimiento de Inicio

Una sesión de diseño con el SE se inicia con la identificación del usuario. Una vez que el usuario ingresa a la sesión, el SE le brinda al usuario instrucciones de como se desarrollará la sesión.

Luego, el SE le consulta al usuario si desea revisar conceptos y definiciones básicas sobre diseño de BD, para lo cual se emplea el tutor del SE.

b) Procedimiento de Estudio de la Organización

El SE le consulta al usuario si puede describir la Organización en función de 4 niveles:

- Area Funcional
- Función
- Proceso
- Actividad

En caso afirmativo, el SE le solicita al usuario que ingrese las Areas Funcionales de la Organización (en caso de que existan).

Por cada Area Funcional de la Organización, el SE le pide al usuario que ingrese las Funciones que desarrolla el Area Funcional.

Después, por cada Función que el usuario ha ingresado, el SE le solicita al usuario que ingrese los principales procesos que se requieren para dicha función.

Finalmente, por cada proceso ingresado el SE solicita al usuario que le indique las principales Actividades que se requieren para realizar dicho proceso.

c) Procedimiento de Definición de Entidades

En este procedimiento, el SE le solicita al usuario que le brinde un conjunto inicial de entidades que se requieren para realizar cada una

de las actividades definidas en el procedimiento anterior.

Además, el SE requiere que por cada entidad se le indique sus atributos y claves candidatas.

d) Procedimiento de Definición de Relaciones

El SE le consulta al usuario si desearia revisar las entidades que ha definido en el paso anterior antes de definir relaciones.

Si es afirmativa su respuesta, el SE le presenta una lista de las entidades definidas.

A continuación el SE le pide al usuario que le ingrese las relaciones entre las entidades y que se los dé en la forma:

(ENTIDAD 1) (VERBO) (ENTIDAD 2)

Además, por cada relación, el SE solicita al usuario que le indique la cardinalidad entre las entidades y los atributos de la relación.

e) Procedimiento de Validación

A partir de la información recolectada en los procedimientos anteriores, el SE realiza una validación de esta información para asegurarse que es consistente y trate de identificar las entidades y verbos que representan la misma información en diferentes formas.

f) Procedimiento de Identificación de Claves Primarias

El SE a partir de las claves candidatas determina en forma automática las claves primarias aplicando sus reglas internas.

g) Procedimiento de Normalización

Finalmente, el SE interactúa con el usuario para identificar y resolver las dependencias funcionales antes de presentar los resultados finales.

4.0 CONCLUSIONES Y RECOMENDACIONES

En el presente documento, hemos presentado un conjunto de criterios y experiencias en el desarrollo de un SE orientado al diseño del modelo conceptual de una BD; a éstos podemos añadir algunas conclusiones y recomendaciones.

4.1 Beneficios del SE

- Permite hacer operacionales las metodologías y técnicas de diseño de una BD.
- Implementa en forma práctica y controlada los conceptos de diseño de una BD e Ingeniería de la Información.
- Brinda facilidades para mejorar la calidad del diseño del modelo conceptual de una BD mediante su verificación en forma automatizada.

- Permite operacionalizar las técnicas de prototipo en el modelamiento de datos.
- Reduce la dependencia del experto diseñador de BD por parte de la Organización.
- Simplifica el mantenimiento del modelo conceptual de la BD.
- Permite reducir el tiempo de desarrollo de un Sistema de Base de Datos en una Organización.
- Brinda facilidades para que el diseñador de BD centre su trabajo en los aspectos creativos del diseño de la BD.
- Reduce los costos de diseño del modelo conceptual de una BD.
- Brinda facilidades para la participación de los usuarios en el diseño del modelo conceptual de una BD, de forma tal que aumente la calidad del diseño y una reducción de errores.

4.3 Desarrollo Futuro del SE

a) El alcance del SE para el diseño del modelo conceptual de una BD, abarca la primera fase en el proceso de diseño de una BD, la orientación en el desarrollo del SE debe dirigirse a la conversión del modelo conceptual al modelo lógico para un SGBD específico y posteriormente la conversión del modelo lógico en un modelo físico.

Además, se debe buscar la generación automática de programas de aplicación que utilizen la BD (con un

SGBD específico) a partir del modelo conceptual de la BD y de las transacciones definidas por los usuarios.

b) Por otro lado, se debe considerar que una mejora importante en el SE sería incrementar su conocimiento general y su habilidad para aplicar razonamientos llevados por el sentido común. Por ejemplo, un diseñador de BD generalmente es capaz de detectar que una entidad es una "persona" (es el caso de un "cliente", implícitamente sabemos que es una "persona").

El reconocimiento de este tipo de casos implica muchas cosas sobre los atributos y relaciones que el diseñador automáticamente puede tomar en cuenta. Además, incrementar la habilidad del SE para realizar este tipo de inferencias permite que la utilización del SE sea menos honerosa, de forma tal que esto permita mejorar considerablemente la calidad de los diseños producidos.

5.0 REFERENCIAS BIBLIOGRAFICAS

1. Borland International Inc.
   "Turbo PROLOG"
   Manual del Producto
   1,986

2. Bouzeghoub M., G. Gardarin y E. Metais
   "Database Design Tools: An expert System Approach"
   Project SABRE
   Universite Paris
   1,985

3. Hawryszkiewycz I. T.
   "A Computer - Aid for E-R Modelling". Proc. 4 TH International Entity - Relationship Conference
   1,985

4. Kawaguchi A., Taoka N., Mizoguchi R., Yamguchi T. y Kakusho O.
   "An Intelligent Interview System for Conceptual Design of Database"
   Proc. ECAI
   1,986

5. Mylopolulos J.
"Knowledge Representation and Databases"
A Panel on AI and Databases Proc. Eighth International Joint Conference on Artificial Intelligence.
1,983

6. Mylopoulos J. y Borgida A., Greenspan S. y Wong H.
"Information System Design at the Conceptual Level - the Taxis Project"
Database Engineering
Volumen 7, Número 4
1,984

7. Navathe S. B. y Elmasri R.
"Integrating User Views in Database Design"
IEEE Computer
1,986

8. Navathe S. B. y Gadgil S. G.
"A Methodology for View Integration in Logical Database Design"
Proc. of the Eighth International Conference on Very Large Data Bases
1,982

9. Storey V. C. y Goldstein R. C.
"A Methodology for the Creation of User Views During Database Design"
ACM Transactions on Database Systems
Volumen 13, Número 3
1,988

10. Storey V. C.
"Knowledge - Based Approaches to Database Design"
University of Rochester
1,989

11. Tauzovich B.
"An Expert System for Conceptual DAta Modelling"
Proc. 8 Th International Conference on Entity - Relationship Approach
1,989

FORUM: UNA HERRAMIENTA PARA LA GESTION DE INFORMACION SOBRE PROBLEMAS, TERMINOLOGIAS Y REFERENCIAS EN TEORIA DE GRAFOS

Carlos Gálvez.
Centro de Ingeniería de Software y Sistemas ISYS. Facultad de Ciencias. Universidad Central de Venezuela. Aptdo. 47002. Los Chaguaramos 1041-A. Caracas. Venezuela.

Francis Losavio.
Escuela de Computación. Universidad Central de Venezuela. Aptdo. 47002. Los Chaguaramos 1041-A. Caracas. Venezuela.

Oscar Meza.
Departamento de Computación y Tecnología de la Información. Universidad Simón Bolívar. Apto. 89000. Caracas. Venezuela.

Oscar Ordaz.
Escuela de Computación. Universidad Central de Venezuela. Aptdo. 47002. Los Chaguaramos 1041-A. Caracas. Venezuela.

RESUMEN

Se presenta FORUM, un sistema interactivo multiventanas para la actualización y la consulta de una base de datos sobre problemas, terminología y referencias bibliográficas en teoría de grafos. Se utiliza la técnica de Hypertexto para la navegación a través de la base de datos.

1. INTRODUCCION.

En este Trabajo[1] se describe el diseño e implementación del sistema FORUM [Gal 90], el cual es una herramienta de gestión de información. FORUM forma parte del AMmbiente interactivo para investigadores en DIgrafos **AMDI** [LMO 90] .

1.1. El ambiente AMDI.

AMDI ofrece al investigador en teoría de digrafos las herramientas gráficas necesarias para manipular digrafos, de la forma

1 Este trabajo forma parte del proyecto AMDI, el cual es financiado por el CDCH y esta siendo desarrollado en el Centro de Investigación ISYS (Ingeniería de Software y Sistemas), de la U.C.V.

más natural y amigable posible, y automatizar las tareas usuales del investigador en dicha área. El ambiente incluye un editor gráfico para la manipulación de dibujos (mediante teclado y ratón), permite calcular parámetros generales y propiedades de un digrafo, permite generar en forma aleatoria digrafos que cumplen o no ciertas propiedades, gestión de archivos MS-DOS, acceso al procesador de palabras WORD 5.0 (el cual tiene la facilidad de importar gráficos) y acceso a FORUM. Finalmente AMDI posee una ayuda en línea. Es de hacer notar que FORUM tiene autonomía, es decir puede funcionar como un sistema aislado, sin necesitar de ninguna de las herramientas de AMDI.

La interfaz-usuario de AMDI está concebida como un sistema de ventanas múltiples, con menúes horizontales y verticales, siguiendo los últimos desarrollos teóricos para el diseño de sistemas gráficos interactivos (Como por ejemplo la filosofía de Macintosh o XWindows [FV 82]). La estructura de los módulos de AMDI sigue la filosofía del informe Stoneman [DOD 80], donde la arquitectura del sistema constituye un esquema jerárquico por niveles. En el nivel 0 denominado núcleo, se localizan todos los dispositivos tanto a nivel de Software como de Hardware, que dependen de la máquina. En el nivel 1 se sitúa la interfaz entre el nivel 0 (máquina), el usuario y el nivel de Herramientas, el cual contiene las herramientas o módulos que constituyen el ambiente y que son fundamentales para su funcionamiento.

1.2. Breve descripción de FORUM.

El objetivo del sistema FORUM es proporcionar información al investigador, generalmente dispersa en varios textos y artículos, sobre complejidad y referencias bibliográficas de problemas clásicos en teoría de grafos, glosarios en Teoría de Grafos, en Complejidad

Algoritmica y en Grafos Aleatorios. Es importante destacar que por su carácter autónomo y la forma como fue diseñado, es posible construir un FORUM sobre otro tópico que no sea teoría de Grafos, para ello basta generar la Base de Datos de problema, glosario y referencias bibliográficas referente al tópico. El sistema fue implementado para microcomputadores PC XT, AT, PS o compatibles , en lenguaje Clipper [Nan 87] con el manejo de "Overlays".

FORUM puede ser visto como un sistema de base de datos. Contiene objetos, relaciones entre ellos, y una serie de operaciones sobre los objetos; estas operaciones se encuentran agrupadas en módulos. La consulta de información en FORUM se realiza aplicando la denominada técnica de HYPERTEXTO [ER 89], [Glu 89], [Ro&al 89], [Ros 89] , [SCG 89]. [AMY 88], la cual es usada hoy en día para la actividad de consulta en aquellos sistemas que manejan grandes cantidades de información. En la figura 1 se muestra la consulta a un problema, donde está resaltado el término CONJUNTO INDEPENDIENTE, que puede ser consultado directamente para saber su significado.

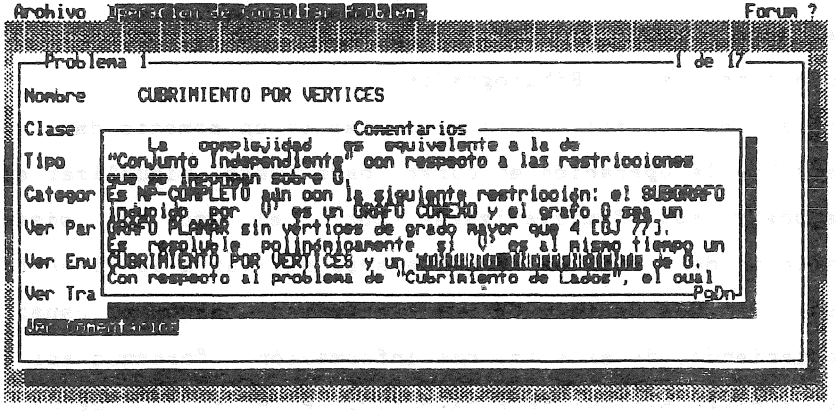


Figura 1 Consulta de un problema

2. EL SISTEMA FORUM.

Siguiendo la misma filosofía del ambiente AMDI, FORUM se comunicará con el usuario mediante un conjunto de ventanas y menúes amigables, ofreciendo la posibilidad de realizar una serie de operaciones sobre la información contenida en: **problema** (estado actual sobre la complejidad algorítmica de problemas clásicos en teoría de grafos [GJ 79]); **glosario** ("Glosario Base para la Teoría de Grafos" [Ort 88], "Complejidad Algorítmica" [BMO 90], y el "Glosario Base para la Teoría de Grafos Aleatorios" [FMO 89]); **bibliografía** (referencias bibliográficas a trabajos relacionados con la complejidad algorítmica de problemas) .

El usuario podrá consultar de una manera sencilla, con la ayuda de la técnica de HYPERTEXTO toda la información antes mencionada. Por ejemplo, supongamos que el usuario se encuentra consultando varios problemas y se sitúa sobre uno en particular, luego al ser desplegada la información referente al problema se tiene la posibilidad de obtener más información, en caso de existir, acerca de Términologías, Referencias Bibliográficas y de otros problemas referenciados , sin la necesidad de que el usuario tenga que interrumpir la consulta del problema para ubicarse dentro de las operaciones de consultar Glosario o consultar la Bibliografía.

También, el usuario podrá consultar en un momento dado la ayuda en línea sobre la operación en curso, para luego visualizar el índice de la ayuda y así familiarizarse con el uso del sistema. El funcionamiento del sistema de ayuda será similar a la filosofía antes mencionada, ya que dentro de una ventana de ayuda el usuario podrá seguir obteniendo, de existir, más información , formando así un árbol de ayuda. Nuestro sistema será extensible en el sentido de permitir al usuario modificar, agregar o eliminar información referente a los

'roblemas, Glosarios y a la Bibliografía. En todo momento el usuario :endrá la posibilidad de imprimir la información desplegada en su)antalla. Asimismo, se permite al usuario configurar FORUM a su gusto, como por ejemplo cambiar de colores, cambiar el fondo de la)antalla, etc. y posteriormente guardar la configuración deseada, para ;ue en una nueva sesión el sistema mantenga la configuración.

A continuación se muestra el diagrama de la herramienta FORUM, :nfocado como un ambiente integrado [Los 89]:

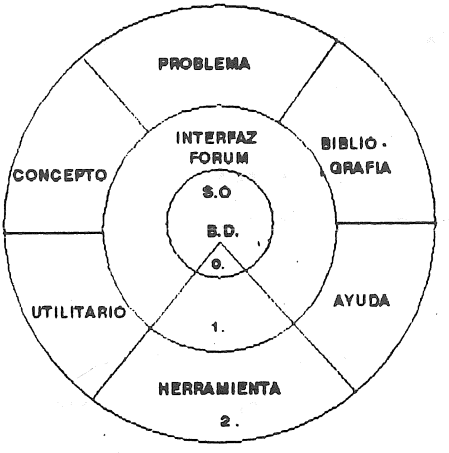


Fig. 2 Diagrama del Módulo FORUM

INTERFAZ-FORUM es el módulo que brinda el acceso a las distintas herramientas y se comunica con el nivel dependiente de la máquina (nivel 0). El nivel HERRAMIENTA contiene los módulos componentes de la Herramienta FORUM: PROBLEMA, CONCEPTO, BIBLIOGRAFIA, AYUDA y UTILITARIO.

El módulo PROBLEMA: Este módulo realiza el manejo de la información de los Problemas clásicos en la Teoría de Grafos y Complejidad Algorítmica [GJ 79], ofreciendo operaciones de Editar, Consultar, Impresión, Inicializar y Reindexar[2] los archivos en donde

[2] En este contexto por indexar entenderemos la asignación de índices y por reindexar entenderemos asignación de nuevos índices.

se encuentra almacenada la información. Dentro de la operación de Editar se podrá modificar o eliminar un problema ya existente, o crear uno nuevo. La operación de Consultar ofrecerá al usuario, primero, la posibilidad de visualizar todos los problemas ordenados alfabéticamente, segundo, obtener todos los problemas por una secuencia de caracteres, la cual forma parte de su nombre, introducidos por el usuario; y por último se pueden consultar los problemas segun sus clases, tipos, categorías y palabras claves.

El módulo CONCEPTO: Este módulo realiza el manejo de la información de los diferentes Glosarios, ofreciendo operaciones de Editar, Consultar, Impresión, Inicializar y Reindexar los archivos en donde se encuentra almacenada la información. Dentro de la operación de Editar se podrá modificar o eliminar un concepto de un Glosario ya existente, o crear uno nuevo, o introducir todo un nuevo concepto. La operación de Consultar ofrecerá al usuario, primero la posibilidad de seleccionar el tipo de Glosario con el cual se desea trabajar (puede ser uno en específico o todos los Glosarios), segundo obtener todos los conceptos en los cuales el usuario introduce una secuencia de caracteres, la cuál forma parte de su término, introducidos por el usuario , tercero visualizar todos los Conceptos en un orden alfabético, y por último visualizar todos los Conceptos en un orden didáctico.

El módulo BIBLIOGRAFIA: Este módulo realiza el manejo de la información acerca de las Referencias Bibliográficas, ofreciendo operaciones de Editar, Consultar, Impresión, Inicializar y Reindexar los archivos en donde se encuentra almacenada la información. Dentro de la operción de Editar se podrá modificar o eliminar una referencia Bibliográfica ya existente, o crear una nueva. La operación de Consultar ofrece al usuario, primero la posibilidad de visualizar

todas las referencias Bibliográficas ordenadas por sus códigos, segundo obtener todas las referencias Bibliográficas en las cuales forma parte de su título una secuencia de caracteres introducidos por el usuario, y por último se pueden consultar las referencias Bibliográficas dado un período de tiempo.

El módulo AYUDA: Este módulo se encarga del manejo de la información referente a la ayuda de la herramienta FORUM, permitiendo que en todo momento el usuario pueda hacer uso de ella visualizando la información.

El módulo UTILITARIO: Este módulo ofrece al usuario una serie de opciones adicionales a las de Problemas, Concepto, Bibliografía y Ayuda. Estas opciones son las de 'RESTORE y BAKCUP' de toda la base de datos, 'OSSHELL' salida al sistema operativo sin perder el ambiente de FORUM, y la opción de CONFIGURACION, la cual permite al usuario configurar el sistema a su gusto, colocando visible o no los comandos que se muestran en la última línea de la pantalla; colocando visible o no la línea de ayuda (esta línea de ayuda es diferente a la ayuda solicitada en un momento dado), en la cual se despliega un pequeño comentario de la operación sobre la cual el usuario se encuentra posicionado en la INTERFAZ-FORUM; cambiando los colores del 'Foreground' y 'Background'; cambiando el fondo de la pantalla y posteriormente guardar la configuración deseada, para que en una nueva sesión, el sistema mantenga la configuración seleccionada.

3. OBJETOS QUE CONFORMAN LA BASE DE DATOS DE FORUM.

FORUM maneja cuatro tipos de entidades, las cuales son: problemas, conceptos, bibliografías y los diferentes tópicos de ayuda, los cuales se incluyen dentro de la base de datos por ser estos

considerados como información adicional que el usuario puede consultar en cualquier momento.

A continuación se describen los objetos junto con sus atributos, entendiéndose por atributos a las características que definen el objeto.

PROBLEMA: Este objeto representa un problema clásico de la teoría de Grafos, sus atributos son:

. CODIGO, viene dado por un número natural (mayor que cero) el cual es asignado automáticamente por el sistema y representa el orden cronológico como han sido introducido los problemas dentro de la base de datos de FORUM.

. NOMBRE, contiene el nombre del problema y viene dado por una cadena de caracteres.

. CLASE, contiene la clase del problema representada por el tipo de Complejidad (NP-Completos, NP-Hard, Polinomiales, Abiertos, etc.)

. TIPO, contiene el tipo del problema (Grafos, Redes, Lógicos, Etc.)

. CATEGORIA, corresponde a una clasificación dentro del tipo (Cubrimientos, Particiones, Subgrafos, Supergrafos, Ordenamiento, Etc.)

. PARAMETROS, o Instancia Genérica de Datos, es un texto que describe los parámetros asociados a un problema.

. ENUNCIADO, contiene el enunciado de las propiedades que debe satisfacer una solución del Problema dado. El enunciado se formula como una pregunta cuya respuesta es SI o NO.

. TRANSFORMACION, consiste en especificar la transformación polinomial que permite obtener el problema en cuestión a partir de otros problemas ya conocidos.

. **COMENTARIOS**, da información sobre la complejidad del problema restringido a ciertas instancias de datos.

CONCEPTO: Este objeto está conformado por conceptos, el conjunto ormado por todos los conceptos pertenecientes a una teoría forman un losario . Los atributos son los siguientes:

. **CODIGO**, es un número natural (mayor que cero) el cual es asignado automáticamente y representa el orden didáctico en el cual ha sido introducido el concepto dentro de la base de datos de FORUM.

. **TIPO**, representa el tipo de Glosario al cual pertenece el concepto, como por ejemplo, Glosario en Teoría de Grafos, Glosario de Grafos Aleatorios, etc.

. **TERMINO**, cadena de caracteres.

. **DEFINICION**, contiene la definición del TERMINO.

BIBLIOGRAFIA: Este objeto está conformado por las referencias ibliográficas en el área de la teoría de Grafos. Los atributos son os siguientes:

. **CODIGO**, es la clave de acceso y es generado automáticamente por el sistema en base a los autores y año de edición de la Bibliografía.

. **AUTORES**, contiene los autores de la Referencia Bibliográfica.

. **TITULO**, contiene el título de la Referencia Bibliográfica.

. **EDITORIAL**, si la Referencia Bibliográfica es un libro contiene la editorial, si es un publicación interna contiene los datos de la publicación.

. **FECHA**, contiene la fecha de publicación y se encuentra formada por el MES y el AÑO.

AYUDA: Este objeto contiene toda la información de ayuda que el usuario pueda requerir en un instante determinado. Los atributos de este objeto son:

. TOPICO, contiene el nombre de la información. Este atributo permite indentificar en forma uníca a un tópico dentro de todo el conjunto que forma la Ayuda de FORUM.

INFORMACION, contiene una descripción detallada del tópico de ayuda.

Relaciones entre los objetos que conforman la base de datos de FORUM:

Los **problemas se relacionan con los conceptos**, esta relación es de uno a varios y quiere decir que un problema puede estar asociado con muchos conceptos del Glosario.

Los **problemas se relacionan con la bibliografía**, esta relación es de varios a varios, ya que un problema está asociado a muchas Bibliografías y viceversa.

Los problemas se relacionan consigo mismo, esta relación es de uno a varios ya que asocia un problema a uno o varios problemas.

Los **conceptos se relacionan consigo mismo**, esta relación es de uno a varios ya que asocia un concepto a uno o varios conceptos.

La **ayuda se relaciona consigo misma**, esta relación es de uno a varios ya que asocia un tópico de la Ayuda a un o varios tópicos.

4. METODOLOGIA DE DISEÑO UTILIZADA.

Se adoptó la misma metodología [FV 82] de diseño utilizada para especificar las otras herramientas de AMDI [Bra 87]. Esta metodología es adecuada para el diseño de sistemas interactivos y consiste en descomponer la fase del diseño en las siguientes etapas: **DISEÑO CONCEPTUAL, DISEÑO SEMANTICO, DISEÑO SINTACTICO y DISEÑO**

LEXICOGRAFICO. En el diseño Lexicográfico construimos el Diagrama de transición de estados, que constituye una representación esquemática del diálogo interactivo que se origina sobre cada una de las herramientas del sistema FORUM [WSH 81].

Ejemplo del uso de la metodología, en el módulo PROBLEMA y el submódulo EDITAR PROBLEMA:

- DISEÑO CONCEPTUAL: Modulo PROBLEMA

OBJETIVO: Tener reunidas lógicamente un conjunto de operaciones definidas sobre el objeto PROBLEMA en EDITAR, CONSULTAR, y ARCHIVO (Inicializar, Reindexar). Esta clasificación se hizo tomando en cuenta la funcionalidad de cada una de las operaciones.

OBJETOS MANEJADOS: Problema (objeto de la base de datos de FORUM).

OPERACIONES SOBRE LOS OBJETOS: Editar Problema, Consultar Problema y Archivo (Inicializar y Reindexar).

Submódulo EDITAR PROBLEMA.

OBJETIVO: Ofrecer al usuario de una manera sencilla y fácil de manejar un conjunto de operaciones de edición sobre el objeto PROBLEMA.

OBJETOS MANEJADOS: Problema (Objeto de la base de datos de FORUM), Ventana y Texto.

OPERACIONES SOBRE LOS OBJETOS:

Problema: Guardar problema, Cargar problema, Eliminar problema, Crear problema, Modificar problema e Imprimir problema.

Ventana: Desplegar ventana de Edición, ventana estática de Edición, ventana dinámica de Edición (EDITOR BASICO). Cerrar ventana de Edición, ventana estática de Edición, ventana dinámica de Edición (EDITOR BASICO).

Texto: Guardar, recuperar, editar.

- DISEÑO SEMANTICO. Módulo PROBLEMA. submódulo EDITAR PROBLEMA

Guardar Problema.

Información requerida: - Todos los objetos de la base de datos.

Resultado: Si el problema, digamos P es nuevo, éste es agregado a los demás problemas de la base de datos de FORUM y el sistema asigna un código en forma automática. Este código representa el orden de creación del problema P. Si el Problema P no es nuevo simplemente se actualiza su información.

- DISEÑO SINTACTICO. Módulo PROBLEMA. submódulo EDITAR PROBLEMA

Esta operación es seleccionada al posicionarse en la opción del menú desplegable de la opción 'Problema' y enviar la señal correspondiente. En este momento y de una forma transparente al usuario, la INTERFAZ-FORUM pasa el control al módulo PROBLEMA. La salida es visualizada sobre la ventana de edición correspondiente al objeto PROBLEMA. El usuario entonces puede comenzar a editar un nuevo problema sobre esta ventana.

A continuación daremos la descripción sintáctica de la operación 'Guardar' de la actividad Editar Problema. Se especifica la entrada y la salida para describir sintácticamente los lenguajes de comunicación Hombre-Máquina.

Guardar problema.

Entrada: Existen dos formas para salvar el problema en curso, ya sea posicionarse sobre la opción 'Guardar' del editor y enviar una señal, o presionando la tecla asignada para guardar problema. Luego se presenta la ventana de Pregunta para verificar Guardar.

Salida: Si la operación fue aceptada se despliega la ventana de mensaje comunicando que el problema actual ha sido salvado.

- DISEÑO LEXICOGRAFICO. Módulo PROBLEMA. submódulo EDITAR

(E) Selección a traves de menú.

(S) Se despliega la ventana de edición correspondiente al objeto PROBLEMA, la cual se muestra en la figura 3.

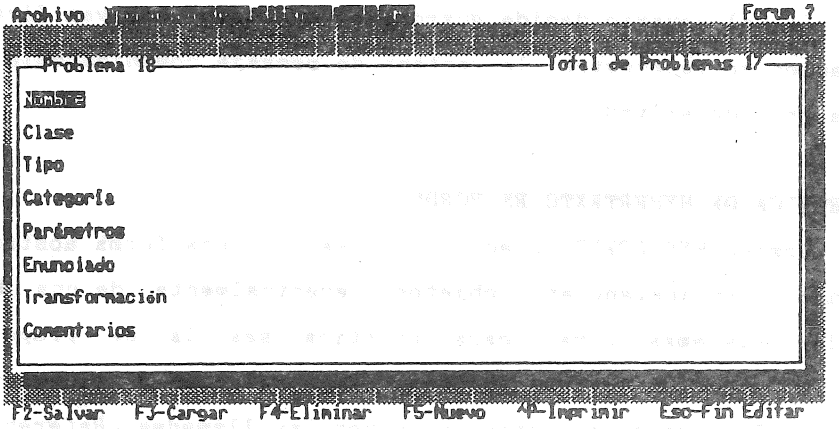


Fig. 3 Pantalla Edición de Problema.

Sobre esta pantalla el usuario puede agregar un nuevo Problema a la base de datos.

Damos a continuación la descripción de la operación **Guardar problema** dentro del módulo 'Editar Problema':

Guardar problema.

(E) Para guardar el problema en curso existen dos formas: la primera es presionando la tecla <<F2>> y la segunda es llevar el cursor hasta la opción 'Guardar', la cual se encuentra en la linea inferior de la pantalla y luego presionar la tecla << Enter >>. Esta operación también es activada si el usuario no ha salvado el problema y decide realizar las actividades de: regresar al menú principal, cargar problema y crear uno nuevo. Posteriormente se presenta la ventana de pregunta donde se le comunica si desea Guardar el problema, no Guardarlo o cancelar la operación de Guardar. Estas tres opciones se muestran en un pequeño menú horizontal dentro de la

ventana de pregunta y la selección de una de ellas se hace posicionándose mediante las teclas <<Flecha Derecha, Flecha Izquierda>> y luego presionar la tecla << Enter >>.

(S) Si el usuario decide guardar el problema, una vez finalizada la operación se le muestra la ventana de mensaje comunicándole que el problema ha sido salvado.

5. LA TECNICA DE HYPERTEXTO EN FORUM.

El término HYPERTEXTO puede definirse como una forma abstracta de relacionar las instancias (objetos), eventualmente de una base de datos de sistemas cuya característica sea la de proporcionar información sobre un determinado tema. Esta relación se representa por los enlaces los cuales son originados por las llamadas (Referencias a otras instancias) que se desean existan sobre cada instancia. En consecuencia un HYPERTEXTO es un digrafo H = (V,E) donde V es el conjunto de nodos que representan las instancias y E son las arcos los cuales representan los enlaces entre dichas instancias. La técnica de HYPERTEXTO consiste en construir el digrafo HYPERTEXTO. Es decir cuales elementos serán considerados y establecer cuales serán las llamadas sobre cada instancia.

En FORUM la técnica de HYPERTEXTO permite al usuario obtener más información, sobre una determinada instancia (PROBLEMA, CONCEPTO, BIBLIOGRAFIA, AYUDA) de la base de datos. Por ejemplo, al visualizar el enunciado de una instancia de problema, el usuario puede obtener la definición de algunos términos (instancia del objeto CONCEPTO), referencias bibliográficas (instancia del objeto BIBLIOGRAFIA) y referencias a otros problemas (instancia del objeto PROBLEMA), que aparecen dentro del mismo.

La técnica del HYPERTEXTO comprende las fases siguientes:

FASE 1 . Consiste en:

(a) la definición de los objetos (nodos) e indicadores (Forma como se señalan las llamadas).

(b) la construcción de las herramientas de **Edición y Consulta**. La herramienta de Edición es utilizada para editar los objetos y colocación de los indicadores sobre determinadas secciones del texto dentro del objeto generando así las llamadas. La herramienta Consulta es utilizada para la activación de las llamadas.

FASE 2. Consiste en hacer uso de la Herramienta Editor para construir un texto con llamadas a partir de los indicadores que previamente se han definido. En esta fase se construye los nodos del digrafo HYPERTEXTO.

FASE 3. Esta fase también denominada de Navegación, consiste en hacer uso de la Herramienta Consultar. La entrada a ésta Herramienta es el texto editado en la FASE 2, el cual es visualizado por ésta herramienta, y al activar una llamada existente dentro del texto, se establece el enlace (arco) entre el objeto representado por el texto editado y el objeto referenciado por la llamada. Luego se visualiza el objeto referenciado, sobre el cual se puede volver a activar cualquier llamada existente sobre el mismo.

Aplicación de las fases a FORUM:

FASE 1.

(a) **definición de los objetos e indicadores**: Los objetos utilizados en FORUM son los de la base de datos Problema, Concepto, Bibliografía y Ayuda. Los indicadores son : COMILLA ("), CORCHETES ([ ]), y la colocación de caracteres en MAYUSCULAS. Cuyo uso quedará establecido en la sección de Reglas de construcción de las llamadas dentro de la FASE 2.

(b) **construcción de las herramientas de Edición y Consulta**: La herramienta de Edición corresponde al EDITOR BASICO del submódulo EDITOR para cada uno de los objetos.

La herramienta de Consulta corresponde al CONSULTOR BASICO del submódulo CONSULTAR para cada uno de los objetos.

FASE 2.

En esta fase decribimos como se construyen las llamadas usando el EDITOR BASICO.

(a) **limitaciones sobre la construcción de llamadas**: Dadas las características (atributos) de los objetos de la base de datos de FORUM, hemos considerado natural imponer las siguientes limitaciones en el uso de la técnica del HYPERTEXTO. Para cada objeto se establece cuales son los atributos donde se colocan las llamadas y los tipos de llamadas permitidas.

A continuación describimos las limitaciones que hemos impuesto para cada objeto :

Para **PROBLEMA**, los atributos donde se realizan llamadas son Parámetros, Enunciado, Transformación y Comentarios. Las llamadas permitidas son a problema, a concepto y a bibliografía. Para **CONCEPTO** el atributo donde se realiza llamadas es Definición. Las llamadas permitidas son a concepto. Para **BIBLIOGRAFIA** el atributo donde se realiza llamadas es Referencias a Problemas. Las llamadas permitidas son a problema. Para **AYUDA** el atributo donde se realiza llamadas es Información. Las llamadas permitidas son a ayuda.

(b) Reglas de construcción de las llamadas.

A PROBLEMA: La llamada a Problema se escribirá colocando su nombre entre comillas. Por tal motivo las comillas son símbolos restringidos para los indicadores a problema.

A CONCEPTO: La llamada a Concepto se escribirá colocando el término, del cual se desea su concepto, en mayúsculas.

A BIBLIOGRAFIA: La llamada a una referencia Bibliográfica se escribirá colocando el código de la referencia Bibliográfica deseada entre corchetes. Por tal motivo los corchetes son símbolos restringidos para los indicadores a referencia bibliográfica.

A AYUDA: La llamada a un Tópico de Ayuda se escribirá colocando el nombre del tópico en mayúsculas.

FASE 3. En esta fase decribimos como se utilizan las llamadas usando el CONSULTOR BASICO para cada objeto. Esta fase se realiza dentro del CONSULTOR BASICO y consiste en pasar de una instancia a otra, eventualmente la misma, a través de los enlaces (arcos) que se originan activando las llamadas existentes dentro del texto que se encuentra visualizado sobre la ventana del CONSULTOR BASICO. Esta acción se repite sobre el texto que representa la nueva instancia hasta, o bien no se desee otra información adicional o se está sobre un texto que no posea llamadas a otras instancias. En estos dos últimos casos, se puede recorrer en sentido inverso el camino (secuencia de nodos y arcos) establecido, hasta un determinado nodo sobre el cual se desea activar otras llamadas que contenga el mismo.

6. CONCLUSIONES.

La técnica de HYPERTEXTO fue implemetada en FORUM sin imitar ningún modelo de paquetes comerciales existentes que utilicen esta técnica. Es de hacer notar que en FORUM, la FASE 2 es relativamente lenta, dado que el sistema para localizar las llamadas en un determinado texto, debe verificar el texto línea a línea y caracter a caracter (se espera incorporar un algoritmo óptimo de búsqueda de patrones de texto). Sin embargo la FASE 3 es sencilla y rápida.

La técnica de HYPERTEXTO puede también usarse para definir procesos de búsqueda de palabras o secuencias de caracteres en un texto. Esta actividad fue también realizada , sin usar HYPERTEXTO, de una manera muy específica en la Consulta, como por ejemplo la búsqueda de un determinado autor dentro de todas la bibliografías existentes en la base de datos.

Puede pensarse en diseñar un FORUM como un sistema que brinde al usuario la posibilidad de construir su propia interfaz para la gestión (edición y consulta) de la base de datos asociada a una determinada área de investigación.

REFERENCIAS.

[AMY 88] Akscyn R., McCracken D. y Yoder E. "KMS : A distributed Hypertext system for managing knowledge in organizations" Comunications of the ACM 31, 7 (Julio 1988), 820-835.

[BMO 90] Brito M. R., Mateo A., Ordaz O. "Un enfoque didáctico sobre la teoría de Complejidad algorítmica ", Reporte No. 1990-4 Dpto. de Matemáticas Puras y Aplicadas, USB, 1990.

[Bra 87] Brazón M. "Primer prototipo del módulo de Edición del Ambiente de trabajo para el investigador en digrafos: AMDI", Tesis de Grado, UCV, 1987.

[DOD 80] Departament of Defense of the USA. "Requirements for the Ada Programing Support Enviroment". Stoneman, Feb. 1980.

[ER 89] Evenson S. y Rheifrank J. "Towards a Design Language for Representing Hypermedia Cues". Hypertext'89 Proceedings, Noviembre 1989.

[FMO 89] Fernandez de la Vega W., Marquez L., Ordaz O. "Un enfoque didáctico sobre la teoría de Grafos Aleatorios". Notas de autor, 1989.

[FV 82] **Foley J.D. y Van Dan A.** "Fundaments of interactive computers graphics". Addison-Wesley, 1982.

[Gal 90] **Gálvez Carlos. "FORUM: una herramienta para la gestión de información sobre problemas, terminologías y referancias en grafos"** Tesis de Grado. U.C.V., 1990.

[GJ 79] **Garey M. y Johnson D.** "Computers and intractability. A guide to the theory of NP-Completeness". Freeman and Campany, 1979.

[Glu 89] **Glushko R.J.** "Design Issues for Multi-Document Hypertexts". Hypertext'89 Proceedings, Noviembre 1989.

[LMO 90] **Losavio F., Meza O. y Ordaz O.** "Ambiente interactivo para digrafos: AMDI". Acta Científica Venezolana. Vol. 41, No.2, 1990.

[Los 89] **Losavio F.** "Towards the constructiond of interactive working environments: A prototype Ada environment". ICNTSSD'89, Caracas, Diciembre 1989.

[Nan 87] "**Clipper, Ver. Summer 87**" Nantucket Corporation, 1987

[Ort 88] **Ortega M.** "Glosario de base para la teoría de Grafos". Por aparecer, Fondo Editorial de Acta Científica.

[Ros 89] DeRose S. J. "Expanding the Notion of links". Hypertext'89 Proceedings, Noviembre 1989

[Ro&al 89] **Rous B., Schlderman B., Yankelovich N. y Yoder E.** "Panel: Lessons learned from the ACM Hypertext on Hypertext Project". Hypertext'89 Proceedings, Noviembre 1989.

[SCG 89] **Shipman F. M., Chaney J. R. y Gorry G. A.** "Distributed Hypertext for Collaborative research: The virtual Noteebook System". Hypertext'89 Proceedings, Noviembre 1989.

[WSH 81] **Wulf W. A., Shaw M., Hilfinger P. N.** "Fundamental Structures of Computer Science". 1981.

CAPACIDADES DE ESPECIFICACION, ANALISIS Y VERIFICACION DE PROPIEDADES DE SUPERVIVENCIA Y VITALIDAD DE SISTEMAS DISTRIBUIDOS SINERGESICOS DEL ESCENARIO SINERGIA_w.

Páez, Charles R. y León, Leandro R.[$]

RESUMEN

La especificación, análisis y verificación de propiedades de supervivencia y de vitalidad de sistemas distribuidos sinergésicos son capacidades principales del escenario SINERGIA_w. El modelo de especificación es una extensión de la red de N MEF comunicantes a través de C canales unidireccionales PEPS contorneados. La técnica de análisis y verificación de la propiedad de progreso infinito utilizada se basa en el producto cartesiano de los N automatas deterministicos. El sistema construye un automata global M_{SDS} al cual identifica cualquier error de diseño que impida la supervivencia del sistema.La técnica de análisis y verificación de la propiedad de realización de servicios infinitamente a menudo se basa en la teoría de lenguajes_w de McNaughton y en el modelo teorico de sistemas distribuidos como una población linguistica_w de C par-sinergones de Páez. En consecuencia, SINERGIA_w desarrolla el modelo matemático del sistema distribuido como un vector H_w de historia infinita del automata global y un matriz L_w de lenguajes_w asociados a cada canal de acciones atómicas sinergésicas que realizan un pase de mensajes asociado a un par-sinergon. Se ilustran las capacidades del escenario usando un convertidor de protocolos de comunicación de Lam.

Palabras Claves: Supervivencia,Vitalidad,Lenguajes_w,Sistemas Distribuidos Sinergésicos.

($) Dirección Presente de Autores : Centro de Estudios en Microcomputacion y Sistemas Digitales CEMISID Facultad de Ingenieria, Universidad de Los Andes, Merida, Venezuela, 5101

I. ESPECIFICACION DE SISTEMAS DISTRIBUIDOS SINERGESICOS

En principio, un **sistema distribuido sinergésico** es concebido como una colección de N **sinergones** M_i donde i=0,1,...,(N-1) y una relación de **sinergia topológica** T entre ellos.

Un **sinergon** M_i es un constructo abstracto de computación en relación de sinergia topológica con otros sinergones M_j en el sistema y se define como una sextupla

$$M_i = \langle S_i, \Sigma_i, \delta_1, sI_i, F_i, P_i \rangle$$

donde S_i es el conjunto finito, no vacio de estados internos, E_i es el alfabeto finito de acciones atómicas sinergésicas en las cuales participa M_i, δ_1 es la función sucesor definida como :

$$\delta_i : S_i \times \Sigma_i \dashrightarrow S_i$$

sI_i es el estado interno inicial, miembro de S_i, F_i es un subconjunto de estados internos finales, F_i S_i y P_i es un vector columna binario (1xN) que especifica la relación de sinergia topológica existente entre el sinergón Mi y los (N-1) sinergones Mj co-existentes.

$$P_i = [\ p_{i0}, \ \cdots\ , p_{ij}, \ \cdots\ , p_{iN}\ ]^T$$

donde cada elemento p_{ij} no-cero indica la existencia de un canal de comunicación c_{ij} asociado a un alfabeto E_{ij} finito de acciones atómicas sinergésicas $G_{ij}:m_k:A_{ij}$ [1] de naturaleza unidireccionales, PEPS, contorneados a una profundidad p de acciones atómicas sinergésicas en cola y libre de error. La

---

(1) G_{ij} es una gramática regular de izquierda a derecha que genera el lenguaje L_u_{ij} sobre $\Sigma_{ij} = \{a_k \mid k = k_1, k_2, \ldots, k_\theta\}$ y A_{ij} es un autómata aceptador de L_u_{ij}. En principio el par-sinergón será indicado mediante $M_i:L_u_{ij}:M_j$

especificación de un elemento p_{ij} cero es indicativo a la ausencia de relación de sinergia topologica entre los sinergones M_i y M_j y por lo tanto de inexistencia de canal de comunicaciones entre ellos.

De este modo, un sistema distribuido sinergésico es especificado como una colección de N sinergones M_i para i=0,1,...,(N-1) y una mátriz NxN de sinergia topológica T :

$$SDS : < M_0, \ldots, M_i, \ldots, M_{N-1}, T >$$

donde $M_0 = <S_0, \Sigma_0, \omega_0, sI_0, F_0, P_0>$

... $M_i = <S_i, \Sigma_i, \omega_i, sI_i, F_i, P_i>$

... $M_{N-1} = <S_{N-1}, \Sigma_{N-1}, \omega_{N-1}, sI_{N-1}, F_{N-1}, P_{N-1}>$

y

$$T = [\ P_0 \ \ldots \ P_i \ \ldots \ P_{N-1}\ ]$$

y C es el numero de elementos no-cero p_{ij} en la matriz T en el rango C : [(N-1) .. $2^{N(N-1)}-1$] y p_{ij} es un entero positivo pequeño.

El escenario SINERGIA_w permite la especificación de un sistema distribuido sinergesico en estos terminos. La Figura 1 ilustra el convertidor de protocolo de enlace de dato IBMBSC a BITALTERNO [LAM-88]. En nuestro modelo el SDS_LAMCP2 consiste de :

$$SDS_LAMCP2 : < M_0, M_1, M_2, T >$$

donde la mátriz de relación de sinergia topológica viene dada por

$$T = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

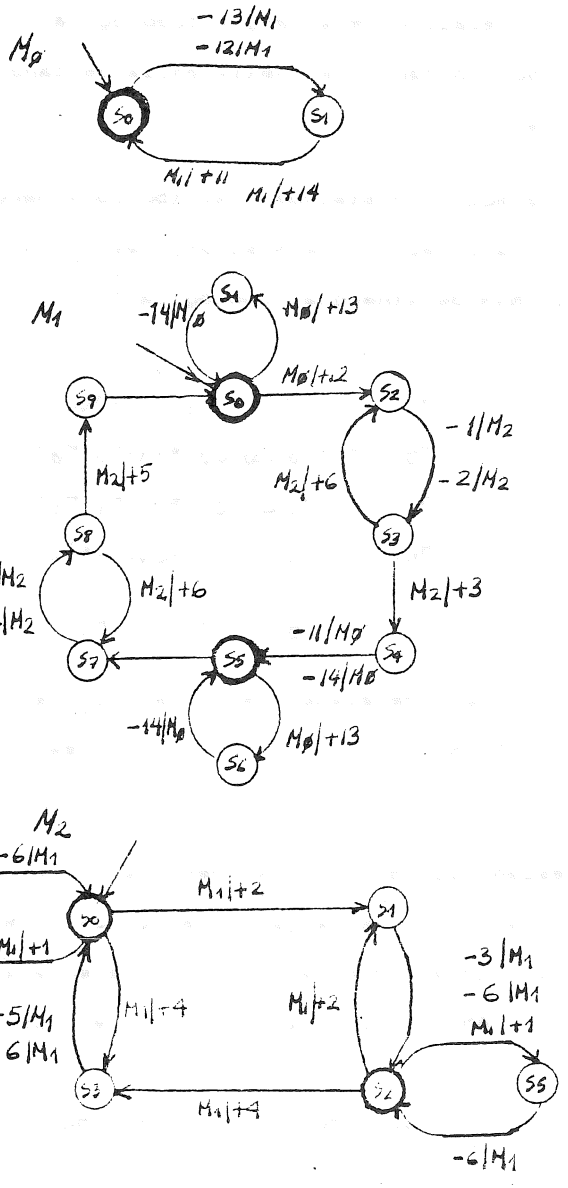


Figura 1 Especificación Convertidor Protocolo de Comunicación IBMBSC a BITALTERNO : SDS-LAMCP2 [LAM-88]

y donde los sinergones M_0, M_1 y M_2 se especifican como :

$M_0 = \langle S_0, \Sigma_0, \delta_0, sI_0, F_0, P_0 \rangle$, donde $S_0 = (s_0, s_1)$, $sI_0 = (s_0)$ $F_0 = (s_0)$, $P_0 = [\ 0,1,0\ ]^T$, $\Sigma_0 = \Sigma_{01} \cup \Sigma_{10} = (M_0|12|M_1, M_0|13|M_1) \cup (M_1|11|M_0, M_1|14|M_0)$.

$M_1 = \langle S_1, \Sigma_1, \delta_1, sI_1, F_1, P_1 \rangle$, donde $S_1 = (s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9)$, $sI_1 = (s_0)$, $F_1 = (s_0, s_5)$, $P_1 = [\ 1,0,1\ ]^T$, $\Sigma_1 = \Sigma_{10} \cup \Sigma_{12} \cup \Sigma_{01} \cup \Sigma_{21} = (M_1|11|M_0, M_1|14|M_0) \cup (M_1|1|M_2, M_1|2|M_2, M_1|4|M_2) \cup (M_0|12|M_1, M_0|13|M_1) \cup (M_2|3|M_1, M_2|5|M_1, M_2|6|M_1)$ y

$M_2 = \langle S_2, \Sigma_2, \delta_2, sI_2, F_2, P_2 \rangle$ donde $S_2 = (\ s_0, s_1, s_2, s_3, s_4, s_5\ )$, $sI_2 = (s_0)$ $F_2 = (s_0, s_2)$, $P_2 = [\ 0,1,0\ ]^T$, $\Sigma_2 = \Sigma_{21} \cup \Sigma_{12} = (M_2|3|M_1, M_2|5|M_1, M_2|6|M_1) \cup (M_1|1|M_2, M_1|2|M_2, M_1|4|M_2)$ y las funciones sucesor δ_0, δ_1 y δ_2 se muestran como diagramas de estado en la Figura 2.

La verificación de la conectividad fuerte entre cada uno de los estados internos de un sinergón M_i se hace mediante la visualización del **arbol de alcance con raiz** sI_i y similarmente la sinergesis de M_i se realiza mediante el cómputo del **arbol de acciones atómicas sinergesicas con raiz silencio** τ. La Figura 3 muestra los arboles asociados al sinergón M_1 que permiten comprobar su conectividad fuerte y la sucesión de acciones atómicas sinergésicas en las cuales está comprometido.

```
Reporte estados Sinergos M1 LAMCP2.SDS

S00  V/F/02/02 :
        S00S01|+13|M0 S00S02|+12|M0
S01  V/T/01/01 :
        S01S00|-14|M0
S02  V/T/02/02 :
        S02S03|-01|M2 S02S03|-02|M2
S03  V/T/02/02 :
        S03S02|+04|M2 S03S04|+03|M2
S04  V/T/02/02 :
        S04S05|-11|M0 S04S05|-14|M0
S05  V/F/02/02 :
        S05S06|+13|M0 S05S07|+12|M0
S06  V/T/01/01 :
        S06S05|-14|M0
S07  V/T/02/02 :
        S07S08|-01|M2 S07S08|-04|M2
S08  V/T/02/02 :
        S08S07|+06|M2 S08S09|+05|M2
S09  V/T/02/02 :
        S09S00|-11|M0 S09S00|-14|M0
```

```
Reporte estados Sinergos M0 LAMCP2.SDS

S00  V/F/02/02 :
        S00S01|-12|M1 S00S01|-13|M1
S01  V/T/02/02 :
        S01S00|+11|M1 S01S00|+14|M1
```

```
Reporte estados Sinergos M2 LAMCP2.SDS

S00  V/F/03/03 :
        S00S04|+01|M1 S00S01|+02|M1 S00S03|+04|M1
S01  V/T/02/02 :
        S01S02|-03|M1 S01S02|-04|M1
S02  V/F/03/03 :
        S02S03|+04|M1 S02S01|+02|M1 S02S05|+01|M1
S03  V/T/02/02 :
        S03S00|-05|M1 S03S00|-06|M1
S04  V/T/01/01 :
        S04S00|-06|M1
S05  V/T/01/01 :
        S05S02|-06|M1
```

Figura 2 Funciones Sucesor $\emptyset$, 1 Y 2

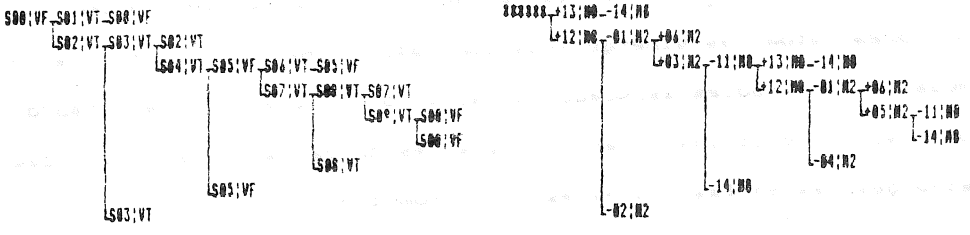


Figura 3. Arbol de Alcance y Arbol de Acciones Atómicas de M_1

II. ANALISIS Y VERIFICACION DE PROPIEDAD DE SUPERVIVENCIA

La técnica sintáctica de análisis y verificación de la propiedad de **supervivencia o progreso infinito** del SDS utiliza el **análisis de perturbaciones** [ZAF-80]. Su objetivo es la verificación de la ausencia de **errores de diseño** que impidan el progreso infinito en la evolución del SDS. Para ello, se adopta una extensión del concepto de **producto cartesiano** de dos **automatas** [RAB-56].

Ello nos permite concebir a un SDS como un **sinergón global** M_{SDS} equivalente a la colección de $M_i = \langle S_i, \Sigma_i, \delta_1, sI_i, F_i, P_i \rangle$ para $i = 0,1,\ldots,N$ sinergones y a la relación de sinergia topológica $T = [P_1, P_2, \ldots, P_n]$ con C elementos no-cero.

Entonces, un **sistema distribuido sinergésico SDS** es definido mediante el producto directo de los N sinergones según T

$$M_{SDS} = M_1 \times M_2 \times \ldots \times M_N$$

que es a su vez un **sinergón global** M_{SDS} definido por

$$M_{SDS} : \langle SS, \Sigma\Sigma, \delta\delta, ssI, FF, T \rangle$$

donde $SS = S_1 \times S_2 \times \ldots \times S_N$ es el conjunto finito de estados internos globales del sistema, $\Sigma\Sigma = \pi(\Sigma_{ij})$ es el alfabeto de acciones atómicas sinergésicas del sistema global que especifica las acciones atómicas de cada uno de los C canales unidireccionales PEPS, contorneados c_{ij}; $ssI = [sI_1, sI_2, \ldots sI_N]$ es el estado inicial del sistema distribuido, $FF = F_1 \times F_2 \times \ldots \times F_N$ es el conjunto de estados

finales del sistema donde todo $ss_f \in FF$ es visitado infinitamente a menudo, $T = [P_1, P_2, \ldots, P_N]$ es la mátriz topológica del sistema con C elementos no-cero y $\delta\delta = \delta_1 \times \delta_2 \times \ldots \times \delta_N$ es la función sucesor del sistema definida como :

$$\delta\delta : SS^x \dashrightarrow SS$$

Dado que, la mátriz topologica del sistema T posee C elementos no-cero ($i <> j$, $p_{ij} <> \emptyset$) , existen C canales de comunicación unidireccionales c_{ij} entre pares de sinergones Mi¦Mj, PEPS, contorneados a una profundidad p de acciones atómicas sinergésicas y libres de error.

SINERGIA_w es cápaz de computar el sinergón global M_{SDS} y presentarlo en forma de **grafo global de sinergia** que es un listado de todos los **estados sinergésicos globales** SS_i asociados al par de la N-tupla de **estados internos de cada sinergón** en el sistemas $s_1 \times \ldots \times s_i \times \ldots \times s_N$ y al C-tupla de **estado de canales** $[c_{\emptyset k}] \ldots [c_{\emptyset p}] \ldots [c_{Nm}]$ del sistema distribuido, donde $P_{\emptyset k}, \ldots, P_{\emptyset p}, \ldots, P_{Nm}$ són todos elementos no-cero en T. A cada estado sinergésico global SS_i se le asocia la tipificación del estado en lo que a progreso infinito se refiere. Mientras sea un **estado vivo** - que admite progreso - o un **estado final** - que realiza un servicio - se dice que el estado garantiza la supervivencia del sistema hacia los **estados sinergesicos globales sucesor** que se listan a continuación. Una segunda alternativa es mostrarlo en forma de **arbol global de sinergia** con raíz **SS0** : sI1,sI2,...,sIN [σ]...[σ]...[σ].

El resultado de este análisis de alcanzabilidad del sistema SDS_LAMCP2 en el escenario SINERGIA_w permite la construcción de

un arbol global de sinergia mostrado en la Figura 4. La información correspondiente a cada estado sinergésico global se muestra en la Tabla 1 donde se lista para cada estado

$$ss_j \; s_{i0}s_{i1}s_{i2} \; [c_{10}][c_{01}][c_{21}][c_{12}] \; \text{EG sucesores-}ss_j$$

donde ss_j es el j^{imo} estado sinergésico global que se asocia a la N-tupla de estados internos $[si_0, si_1, si_2]$ de cada uno de los sinergones donde $si_0 \in S_0$, $si_1 \in S_1$ y $si_2 \in S_2$ y al estado de cada uno de los c canales del sistema $[c_{10}][c_{01}][c_{21}][c_{12}]$ asociados a los elementos $p_{ij} <> 0$ en T. EG define la propiedad de supervivencia exhibida por el estado global dado que EG = {EI,EV,EM,EF,AI,NE} donde EV indica estado vivo, EI estado inicial, EF estado final ($ss_j \in FF$), EM estado muerto, AI estado con acción inesperada, y NE estado n-estable.

En consecuencia, un sistema distribuido sinergésico D exhibe progreso infinito si y solo si EI es un conjunto unitario, EF ≠ {Ø}, EM = {Ø}, AI = {Ø} y todo ss_j E EV ∪ EF.

La Figura 5 muestra las capacidades de análisis y verificación de la propiedad de supervivencia para el protocolo-caso de Zafiropulo et.al. [ZAF-80] en el cual se identifican situaciones multiples de saturación de canales, estados muertos, recepciones inesperadas, N-tuplas estables, ambiguedades de estados globales e iteraciones no-ejecutables.

TABLA 1 ESTADOS SINERGESICOS GLOBALES DE SDS_LAMCP2

```
SS0000 S00S00S00 EF [00][00][00][00] M0/-12/M1;SS0001 M0/-13/M1;S00036
SS0001 S01S00S00 EV [12][00][00][00] M0/+12/M1;SS0002
SS0002 S01S02S00 EV [00][00][00][00] M1/-01/M2;SS0003 M1/-02/M2;SS0004
SS0003 S01S03S00 EV [00][00][01][00] M1/+01/M2;SS0004
SS0004 S01S03S04 EV [00][00][00][00] M2/-06/M1;SS0005
SS0005 S01S03S00 EV [00][00][00][06] M2/+06/M1;SS0002
SS0006 S01S03S00 EV [00][00][02][00] M1/+02/M2;SS0007
SS0007 S01S03S01 EV [00][00][00][00] M2/-03/M1;SS0008 M2/-06/M1;SS0031
SS0008 S01S03S02 EV [00][00][00][03] M2/+03/M1;SS0009
SS0009 S01S04S02 EV [00][00][00][00] M1/-11/M0;SS0010 M1/-14/M0;SS0030
SS0010 S01S05S02 EV [00][11][00][00] M1/+11/M0;SS0011
SS0011 S00S03S02 EF [00][00][00][00] M0/-12/M1;SS0012 M0/-13/M1;S00020
SS0012 S01S03S02 EV [12][00][00][00] M0/+12/M1;SS0013
SS0013 S01S07S02 EV [00][00][00][00] M1/-01/M2;SS0014 M1/-04/M2;SS0017
SS0014 S01S08S02 EV [00][00][01][00] M1/+01/M2;SS0015
SS0015 S01S08S05 EV [00][00][00][00] M2/-06/M1;SS0016
SS0016 S01S08S02 EV [00][00][00][06] M2/+06/M1;SS0013
SS0017 S01S08S02 EV [00][00][04][00] M1/+04/M2;SS0018
SS0018 S01S08S03 EV [06][00][00][00] M2/-05/M1;SS0019 M2/-06/M1;SS0023
SS0019 S01S08S00 EV [00][00][00][05] M2/+05/M1;SS0020
SS0020 S01S09S00 EV [00][00][00][00] M1/-11/M0;SS0021 M1/-14/M0;SS0022
SS0021 S01S00S00 EV [00][11][00][00] M1/+11/M0;SS0000
SS0022 S01S00S00 EV [00][14][00][00] M1/+14/M0;SS0000
SS0023 S01S08S00 EV [00][00][00][06] M2/+06/M1;SS0024
SS0024 S01S07S00 EV [00][00][00][00] M1/-01/M2;SS0025 M1/-04/M2;SS0027
SS0025 S01S06S00 EV [00][00][01][00] M1/+01/M2;SS0026
SS0026 S01S09S04 EV [00][00][00][00] M2/-06/M1;SS0023
SS0027 S01S00S00 EV [00][00][04][00] M1/+04/M2;SS0018
SS0028 S01S05S02 EV [13][00][00][00] M0/+13/M1;SS0029
SS0029 S01S04S02 EV [00][00][00][00] M1/-14/M0;SS0030
SS0030 S01S05S02 EV [00][14][00][00] M1/+14/M0;SS0011
SS0031 S01S03S02 EV [00][00][00][06] M2/+06/M1;SS0032
SS0032 S01S02S02 EV [00][00][00][00] M1/-01/M2;SS0033 M1/-02/M2;SS0035
SS0033 S01S03S02 EV [00][00][01][00] M1/+01/M2;SS0034
SS0034 S01S03S05 EV [00][00][00][00] M2/-06/M1;SS0031
SS0035 S01S03S02 EV [00][00][02][00] M1/+02/M2;SS0007
SS0036 S01S00S00 EV [13][00][00][00] M0/+13/M1;SS0037
SS0037 S01S01S00 EV [00][00][00][00] M1/-14/M0;S00022
```

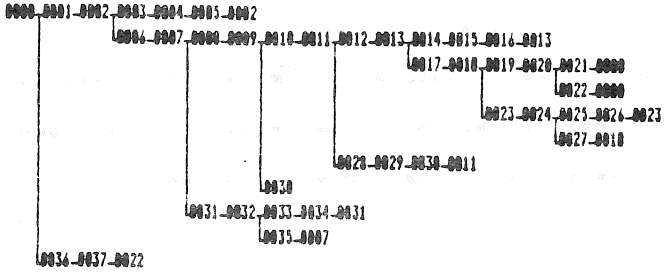


Figura 4 Arbol Global de Sinergia de SDS_LAMCP2

```
SS0000 S00S00 EV [00 00 00][00 00 00] H0/-01/H1;SS0001 H1/-03/H0;SS0014
SS0001 S01S00 EV [01 00 00][00 00 00] H0/+01/H1;SS0002 H0/-02/H1;SS0009 H1/-03/H0;SS0017
SS0002 S01S02 EV [00 00 00][00 00 00] H0/-02/H1;SS0003
SS0003 S02S02 EV [02 00 00][00 00 00] H0/+02/H1;SS0004
SS0004 S02S00 EV [00 00 00][00 00 00] H1/-03/H0;SS0005
SS0005 S02S01 EV [00 00 00][03 00 00] H1/+03/H0;SS0006
SS0006 S00S01 EV [00 00 00][00 00 00] H0/-01/H1;SS0007
SS0007 S01S01 EV [01 00 00][00 00 00] H0/+01/H1;SS0002 H0/-02/H1;SS0000
SS0008 S02S01 EV [01 02 00][00 00 00] H0/+01/H1;SS0003
SS0009 S02S00 EV [01 02 00][00 00 00] H0/+01/H1;SS0003 H1/-03/H0;SS0010
SS0010 S02S01 EV [01 02 00][03 00 00] H0/+01/H1;SS0011 H1/+03/H0;SS0024
SS0011 S02S02 EV [02 00 00][03 00 00] H0/+02/H1;SS0012 H1/+03/H0;SS0021
SS0012 S02S00 EV [00 00 00][03 00 00] H1/+03/H0;SS0000 H1/-03/H0;SS0013
SS0013 S02S01 EV [00 00 00][03 03 00] H1/+03/H0;SS0014
SS0014 S00S01 EV [00 00 00][03 00 00] H1/+03/H0;SS0015 H0/-01/H1;SS0017
SS0015 S01S01 EV [00 00 00][00 00 00] H0/-02/H1;SS0016
SS0016 S02S01 RE [02 00 00][00 00 00]
SS0017 S01S01 EV [01 00 00][03 00 00] H0/+01/H1;SS0018 H1/+03/H0;SS0020 H0/-02/H1;SS0010
SS0018 S01S02 EV [00 00 00][03 00 00] H1/+03/H0;SS0019 H0/-02/H1;SS0011
SS0019 S02S02 EN [00 00 00][00 00 00]
SS0020 S02S01 EV [01 00 00][00 00 00] H0/+01/H1;SS0019
SS0021 S00S02 EV [02 00 00][00 00 00] H0/+02/H1;SS0000 H0/-01/H1;SS0022
SS0022 S01S02 EV [02 01 00][00 00 00] H0/+02/H1;SS0001 H0/-02/H1;SS0023
SS0023 S02S02 EV [02 01 02][00 00 00] H0/+02/H1;SS0009
SS0024 S00S01 EV [01 02 00][00 00 00] H0/+01/H1;SS0021 H0/-01/H1;SS0025
SS0025 S01S01 EV [01 02 01][00 00 00] H0/+01/H1;SS0022 H0/-02/H1;SS0026
SS0026 S02S01 SC [01 02 02][00 00 00]
```

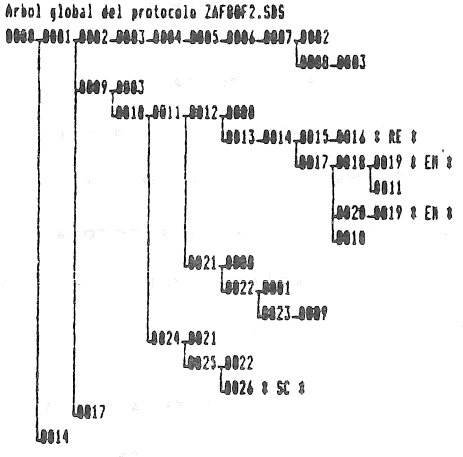


Figura 5 Verificación Supervivencia Protocolo-Caso Zafiropulo et.al. [ZAF-80]

III. ANALISIS Y VERIFICACION DE PROPIEDAD DE VITALIDAD

Cuando el sistema distribuido es **inmortal**, es decir exhibe la propiedad de supervivencia, es posible una tercera alternativa de formalización de su conducta a través de la representación sintáctica de la **historia infinita del sistema distribuido sinergésico SDS**. Esta propiedad puede ser descrita mediante la historia infinita de estados globales HSS que es un lenguaje_w regular H_w sobre el monoide SS^* y por la colección de los C lenguajes_w regulares L_w_{ij} sobre Σ_{ij}^* **producidos/aceptados** por cada par sinergon M_i:M_j, donde M_j es miembro de P_i . Tales lenguajes_w L_w_{ij} resumen la historia infinita de las secuencias de acciones atómicas sinergésicas -pase de mensajes - que se mueven a través de cada canal de comunicación c_{ij}, correspondiente al elemento $p_{ij} <> 0$ de la mátriz T de sinergia topológica del sistema distribuido.

$$SDS : \langle H_w, L_w \rangle$$

Esta mátriz L_w de lenguajes L_w_{ij} es un **observable**. En consecuencia es un mecanismo sintáctico de caracteristicas especiales para formalizar la conducta infinita de un sistema distribuido sinergésico, donde la **sinergia** del sistema es completamente determinada. Mientras que, la historia infinita de estados sinergésicos no lo es.

Es aqui donde nuestra propuesta teórica es la de concebir todo sistema distribuido sinergésico como una **población** linguistica de C par-sinergon M_i:L_w_{ij}:M_j , cada uno de ellos asociado a un canal de comunicaciones c_{ij} unidireccional, PEPS,

contorneado a una profundidad p de acciones atómicas sinergésicas y libres de error a través del cual se sucede entre ellos una comunicación sinergésica expresada en términos de un lenguaje_w regular L_w_{ij} en el sentido de McNaughton [McN-66].

El escenario SINERGIA_w posee una concha de evaluación sintáctica de expresiones regulares [McN-82] y eventos_w [McN-66]. Esta capacidad es utilizada para determinar las formalizaciones de la propiedad de vitalidad de un SDS.

De este modo, es posible en el escenario SINERGIA_w computar el lenguaje_w H_w sobre SS^* transitado por el grafo global de sinergia, expresado como :

$$H_w = (\alpha_1 \cdot \beta_1{}^w + \ldots + \alpha_{\#(FF)} \cdot \beta_{\#(FF)}{}^w)$$

donde α_j es una expresión regular y $\beta_j{}^w$ es un evento_w y #(FF) es el número de estados globales finales SS_f FF.

Asi, el protocolo SDS_LAMCP2 exhibe una historia infinita de estados sinergésicos globales expresado por (1) :

```
0.(((36.37.22+(1+1.(2.3.4.5)^).2.(6+6.7.(31+(31.32.33.34)^.31).32.35.((7+7.(31.32.33.34)^).31.32.35)0).7.8.9.(10+30).11.
(28.29.30.11)0.(12+12.(13.14.15.16)^).13.(17+17.18.(23+(23.24.25.26)^.23).24.27.((18+18.(23.24.25.26)^).23.24.27)0).16.1
9.20.(21+22)).0)^+(37.22)0.(1+1.(2.3.4.5)^).2.(6+6.7.(31+(31.32.33.34)^.31).32.35.((7+7.(31.32.33.34)^).31.32.35)0).7.8.
9.(10+30).11.((28.29.30+(12+12.(13.14.15.16)^).13.(17+17.18.(23+(23.24.25.26)^.23).24.27.((18+18.(23.24.25.26)^).23.24.2
7)0).18.19.20.(21+22).0.(37.22)0.(1+1.(2.3.4.5)^).2.(6+6.7.(31+(31.32.33.34)^.31).32.35.((7+7.(31.32.33.34)^).31.32.35)0
).7.8.9.(10+30)).11)^)
```

---

(1) El simbolo ^ equivale al cierre de la primera ordinalidad infinita w y el * al cierre de Kleene.
El operador unión es el simbolo +; mientras que, el punto . es el operador de concatenación.

Ahora bien, la tarea de verificación de la propiedad de vitalidad de un sistema distribuido inmortal debe formalizar la conducta infinita del sistema exhibiendo la propiedad de alcance de estados globales sinergésicos finales $SS_f \in FF$ de servicio infinitamente a menudo.

De este modo el sistema ya inmortal exhibe una vitalidad tal que garantiza ser util - realización de servicios - infinitamente a menudo sin necesidad de utilizar lógica temporal.

Para ello, se propone la determinación de un grafo sintáctico de servicios que se obtiene mediante el algoritmo grafo-a-expresión [McN-82] que sistematicamente elimina del grafo global de sinergia uno a uno de sus estados ss_k tales que $ss_k \notin ssI$ y $ss_k \notin FF$. De este modo el grafo sintáctico de servicios es un grafo global de sinergia reducido al conjunto de estados sinergésicos globales en (ssI) ∪ (FF) y donde cada arco en el grafo desde un estado ss_t a un estado ss_f, donde $ss_t, ss_f \in$ (ssI) ∪ (FF), esta etiqueteado por una expresión regular r_{tf} sobre $\Sigma\Sigma^*$. Evidentemente que, asociado a cada expresión regular r_{tf} sobre $\Sigma\Sigma^*$ existe a manera de isomorfismo una expresión regular h_{tf} sobre SS^*.

El escenario SINERGIA_w exhibe la capacidad de determinar el grafo sintáctico de servicios y cada una de las expresiones regulares h_{tf} asociadas a cada transición de estados globales en el grafo y cada una de las expresiones regulares r_{tf} correspondiente

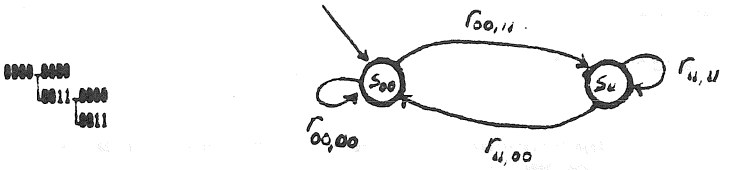


Figura 6 Grafo Sintáctico de Servicios de SDS_LAMCP2

a cada uno de los C canales de la población linguistica de parsinergón.

Asi, la Figura 6 muestra el grafo sintáctico de servicios del SDS_LAMCP2. Y a continuación se muestran los resultados del etiquetamiento asociado a cada uno de las transiciones de estado en el grafo sintáctico de servicio en lo relativo a expresiones regulares de historias de sucesores h_{tf}

```
Reporte Historias Sucesores
H(0000,0000) = 0
36.37.22.0
H(0000,0011) = 0
(1+1.(2.3.4.5)^).2.(6+6.7.(31+(31.32.33.34)^.31).32.35.((7+7.(31.32.33.34)^).31.32.35)0).7.8.9.(10+30).11
H(0011,0000) = 11
(12+12.(13.14.15.16)^).13.(17+17.18.(23+(23.24.25.26)^.23).24.27.((18+18.(23.24.25.26)^).23.24.27)0).10.19.20.(21+22).0
H(0011,0011) = 11
28.29.30.11
```

como de expresiones regulares de historia de acciones atómicas sinergésicas r_{tf} en cada uno de los C canales asociados a cada par-sinergón $M_i:M_j$.

```
Reporte Expresiones Regulares Globales. Canal M0M1. Profundidad 0 LAMCP2.SDS
R(0000,0000) = 6
13.6
R(0000,0011) = 0
12.6
R(0011,0000) = 11
12.6
R(0011,0011) = 11
13.6
```

```
Reporte Expresiones Regulares Globales. Canal M1M0. Profundidad 0 LAMCP2.SDS
R(0000,0000) = 0
6.14.6
R(0000,0011) = 0
11+14+6.((14.6+11+14+6.14.6)
R(0011,0000) = 11
6.((11+14).6+11+14+6.14.6)
R(0011,0011) = 11
6.14.6
```

```
Reporte Expresiones Regulares Globales. Canal M1M2. Profundidad 0 LAMCP2.SDS
R(0000,0000) = 6
6
R(0000,0011) = 0
(6+(6.1.6)^).6.2.(6+(6+(6.1.6)^).2.((6+(6.1.6)^).6.2)0).6
R(0011,0000) = 11
(6+(6.1.6)^).6.4.(6+(6+(6.1.6)^).4.((6+(6.1.6)^).6.4)0).6
R(0011,0011) = 11
6
```

```
Reporte Expresiones Regulares Globales. Canal M2M1. Profundidad 0 LAMCP2.SDS
R(0000,0000) = 0
6
R(0000,0011) = 0
(6+(6.6)^+(6+(6.6)^).6.(6.6)^.(6.(6.6)^)0).6.3.6
R(0011,0000) = 11
(6+(6.6)^+(6+(6.6)^).6.(6.6)^.(6.(6.6)^)0).6.5.6
R(0011,0011) = 11
6
```

En conclusión, el escenario SINERGIA_w implementa el cómputo de toda la propuesta de Páez [PAE-91] de concebir todo sistema distribuido sinergésico como una **población linguística de C par-sinergón** $M_i:L_w_{ij}:M_j$ definida estructuralmente por la relación de sinergia topológica T, cada uno de ellos esta asociado a un canal de comunicaciones c_{ij} unidireccional, PEPS, contorneado a una profundidad p de acciones atómicas sinergésicas y libres de error a través del cual se sucede entre ellos una comunicación sinergésica expresada en términos de un lenguaje_w regular L_w_{ij} en el sentido de McNaughton [McN-66].

SDS : <H_w, L_w>

La conducta en la primera ordinalidad infinita w de cada uno de los C canales c_{ij} asociados a la actividad cooperante del par-sinergón $M_i:L_w_{ij}:M_j$ se expresa mediante la matriz L_w de C elementos no-cero correspondiente a cada uno de los C lenguajes L_W_{ij} asociados a cada canal c_{ij} en el sistema distribuido a través de los cuales se sucede la sucesión de acciones atómicas sinergésicas que realizan el objetivo global del sistema distribuido.

Los elementos no-cero de la matriz L_w correspondiente al sistema SDS_LAMCP2 son los siguientes :

Reporte Expresion Regular Global. Canal M001. Profundidad 0 LAMCP2.SDS
(e+(13.e)*.12).e.((13+12.e.(13.e)*.12).e)*

Reporte Expresion Regular Global. Canal H1H0. Profundidad 0 LAMCP2.SDS

(ε.14.6+(11+14+ε.(14.6+11+14+ε.14.6)).(ε.14.6)8.ε.((11+14).6+11+14+ε.14.6))*9(ε.14.6)8.(11+14+ε.(14.6+11+14+ε.14.6)).(ε.(14.6+((11+14).6+11+14+ε.14.6).(ε.14.6)8.(11+14+ε.(14.6+11+14+ε.14.6))))*

Reporte Expresion Regular Global. Canal H1H2. Profundidad 0 LAMCP2.SDS

((6+(6+(6.1.6)^).6.2.(6+(6+(6.1.6)^).2.((6+(6.1.6)^).6.2)8).6.(6+(6.1.6)^).6.4.(6+(6+(6.1.6)^).4.((6+(6.1.6)^).6.4)8)).6)*+6.(6+(6.1.6)^).6.2.(6+(6+(6.1.6)^).2.((6+(6.1.6)^).6.2)8).6.((6+(6+(6.1.6)^).6.4.(6+(6+(6.1.6)^).4.((6+(6.1.6)^).6.4)8).6.(6+(6.1.6)^).6.2.(6+(6+(6.1.6)^).2.((6+(6.1.6)^).6.2)8)).6)*

Reporte Expresion Regular Global. Canal H2H1. Profundidad 0 LAMCP2.SDS

((6+(6+(6.6)^+(6+(6.6)^).6.(6.6)^.(6.(6.6)^)8).6.3.6.(6+(6.6)^+(6+(6.6)^).6.(6.6)^.(6.(6.6)^)8).6.5).6)*+6.(6+(6.6)^+(6+(6.6)^).6.(6.6)^.(6.(6.6)^)8).6.3.6.((6+(6+(6.6)^+(6+(6.6)^).6.(6.6)^.(6.(6.6)^)8).6.5.6.(6+(6.6)^+(6+(6.6)^).6.(6.6)^.(6.(6.6)^)8).6.3).6)*

IV CONCLUSIONES

Se presenta un experimento que ejercita las capacidades de una herramienta de investigación en el área de sistemas distribuidos y redes de computadoras. El escenario automatizado SINERGIA_w v.2.0. [LEO-91][ACO-89] implementa las ideas de una propuesta teórica de concepción de sistemas distribuidos [PAE-91] como una población linguistica de C par-sinergones determinado por una relación de sinergia topológica T. El sistema exhibe capacidades de especificación de sistemas distribuidos sinergesicos como una extensión del modelo de red de N MEF comunicantes a través de C canales unidireccionales PEPS contorneados. La técnica de análisis y verificación de la propiedad de progreso infinito utilizada se basa en el producto cartesiano de los N automatas deterministicos. El sistema construye un autómata global M_{SDS} al cual identifica cualquier error de diseño que impida la

supervivencia del sistema.La técnica de análisis y verificación de la propiedad de realización de servicios infinitamente a menudo se basa en la teoría de lenguajes_w de McNaughton y en el modelo teórico de sistemas distribuidos como una población linguistica_w de C par-sinergones [PAE-91]. En consecuencia, SINERGIA_w desarrolla el modelo matemático original de un sistema distribuido como un vector H_w de historia infinita del automata global y una matriz L_w de lenguajes_w asociados a cada canal de pase de mensajes de un par-sinergon derivados del grafo sintáctico de servicios del SDS. Las capacidades del escenario se ilustraron usando un convertidor de protocolos [LAM-88].

V REFERENCIAS

[ACO-89] Acosta,J.L., "SINERGIA: Sistema programado de análisis y verificación de propiedades de supervivencia de protocolos de comunicación", Reporte Técnico CEMISID-89-02, Mérida; Universidad de Los Andes, [1989]; 64 p.

[AHU-75] Aho, A.V., Hopcroft, J.E. y Ullman, J.D., et. al. The design and analysis of computer algorithms. Reading, Massachusetts: Addison-Wesley Publishing Company, [1975]; 470 p.

[AHU-83] Aho, A.V., Hopcroft, J.E. y Ullman, J.D., et. al. Data structures and algorithms. Reading, Massachusetts: Addison-Wesley Publishing Company, [1983]; 427 p.

[CAD-89] Carroll, J. y Darrell, L., Theory of finite automata with an introduction to formal languages. New Jersey: Prentice-Hall International, Inc. [1989]; 438 p.

[DAN-80] Danthine, A.A. "Protocol representation with finite-state models". IEEE Transaction on Communications. Vol. COM-28, No. 4, pp. 632-642, April [1980].

[LAM-88] Lam, S. "Protocol Conversion". IEEE Transaction on Software Engineering. Vol. 14, No. 3, pp. 353-361, March [1988].

[LEO-90] León, L.R. "SINERGIA_w Hacia la conceptualización de un escenario automatizado de análisis de propiedades de supervivencia y vitalidad de sistemas distribuidos

sinergésicos". Reporte Técnico CEMISID-91-01, Mérida; Universidad de Los Andes, [1991]; 163 p.

[McN-66] McNaughton, R., "Testing and Generating Infinite Sequences by a Finite Automaton". Information and Control. Vol. 9, pp. 521-530, [1966].

[McN-82] McNaughton, R., Elementary computability, formal languages and automata. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., [1988]; 168 p.

[PAE-91] Páez, C.R., "A Syntactic model for Synergesic Distributed Systems". Sometido a publicación. [1991]

[SAB-88] Sabnani, K. y Anton, D., "A Protocol Test Generation Procedure". Computer Network and ISDN Systems. Vol. 15, pp. 285-297, [1988].

[ZAF-80] Zafiropulo, P., West, C., Rudin, H. et.al. "Towards Analysing and Synthesizing Protocols". IEEE Transactions on Communication. Vol. COM-28, No. 4, pp. 651-660, April [1980].

ASTUTO - UMA ARQUITETURA PARA O DESENVOLVIMENTO DE SISTEMAS BLACKBOARD EM PROLOG

GILBERTO SHIGUEO NAKAMITI
Universidade Estadual de Campinas
UNICAMP/FEE/DCA
Rua Marechal Deodoro, 906 apto 123 - Centro
13020 - Campinas - S.P. - Brasil

RESUMO

Blackboard é um paradigma de resolução de problemas em Inteligência Artificial, capaz de utilizar várias fontes de conhecimento de naturezas diferentes, que agem oportunisticamente para a resolução de um problema.

ASTUTO (*A*rquitetura para a Solução de problemas a*T*ravés do *U*so de *T*écnicas de IA e *O*portunismo) é uma arquitetura de desenvolvimento de Sistemas Blackboard, que dá ênfase às características relacionais entre as entidades do blackboard. Em ASTUTO, tentamos retomar a essência do modelo blackboard, apresentado inicialmente por Newell [07] e discutido em [04]. Para isso, combinamos uma modelagem do blackboard através do Modelo Entidade-Relacionamento (MER) [01] e uma abordagem dirigida por eventos.

PALAVRAS-CHAVE - blackboard, modelo entidade-relacionamento, oportunismo, prolog.

1. INTRODUÇÃO

Um Sistema Blackboard tradicional é constituído por fontes de conhecimento, por um mecanismo de controle, e pelo blackboard propriamente dito. O domínio do conhecimento necessário para a solução do problema é particionado entre as fontes de conhecimento, que são mantidas separadas e independentes, com o intuito de que levem à solução através da informação provida. O blackboard nada mais é que uma base de dados, que consiste de objetos do espaço de soluções. Seu objetivo é servir como meio através do qual as fontes de conhecimento interagem e conduzem à formação incremental e oportunística da solução do problema. O mecanismo de controle disciplina a execução das fontes de conhecimento, decidindo quais caminhos tomar, e monitora o acesso ao blackboard.

As fontes de conhecimento também têm papel muito importante no Modelo Blackboard. Elas devem agir oportunística e cooperativamente, acrescentando informação e conectando as ilhas de solução. Devem ainda ter acesso ao blackboard sob diferentes níveis de abstração, para que esse processo possa ser realizado de forma mais eficaz. Desta forma, o modelamento do blackboard deve ser flexível e permitir representações diferentes da solução sob níveis diferentes de abstração, encapsulando detalhes em entidades de maior poder semântico.

ASTUTO é um sistema que utiliza o Modelo Entidade-Relacionamento para modelar e implementar a estrutura do blackboard, e a linguagem Prolog para sua implementação. Na seção dois, é apresentada uma visão geral de ASTUTO e de seu funcionamento. A seção três apresenta o modelamento e implementação do blackboard Na seção quatro, é mostrado um exemplo de como as Fontes de Conhecimento atuam sobre o blackboard.

2. VISÃO GERAL DO SISTEMA

O objetivo do sistema ASTUTO é se constituir em uma ferramenta através da qual se possa desenvolver e experimentar técnicas e teorias no domínio da aplicação. Pretende-se propiciar a obtenção de uma melhor compreensão do domínio através de seu modelamento, e facilitar o entendimento dos princípios de resolução de problemas nele aplicáveis. O desempenho do sistema não foi fator importante na concepção de ASTUTO, mas sua versatilidade quanto ao domínio da aplicação, modularidade e facilidade de manutenção e extensão do sistema.

A analogia do modelo blackboard com a montagem do quebra-cabeças, presente em [05], nos traz muito de sua essência e do oportunismo intrínseco. As pessoas devem ter visão global e detalhada do estado da solução para que melhor possam contribuir. Além disso, alguns acontecimentos podem alterar significativamente sua visão

do problema e caminhos de raciocínio. Isso pode ocorrer, por exemplo, quando duas ou mais porções do quebra-cabeças (ilhas de solução) se unem em uma única. É interessante que as Fontes de Conhecimento possam ser alertadas sobre fatos relevantes, para que possam se adequar mais rapidamente ao estado atual do problema. Um mecanismo de controle dirigido por eventos pode viabilizar a reprodução desse tipo de comportamento.

Para obter um comportamento dirigido por eventos, foi utilizada a metodologia SADT- Structured Analysis and Design Technique, apresentada em [11]. Através dela, obtivemos uma forte associação entre eventos e ações, permitindo que certos tipos de eventos possam alterar o raciocínio do sistema, e que outros tipos somente modifiquem o blackboard.

Foi também bastante utilizado o princípio de ocultamento de informação, bastante discutido na literatura (por exemplo, em [08,09]), e apresentado em [03] como recurso importante para se obter, por exemplo, características de flexibilidade em sistemas blackboard.

Na figura 1, observamos um esquema conceitual do funcionamento de sistemas desenvolvidos a partir de ASTUTO. Nele, as fontes de conhecimento atuam através de primitivas de vários tipos. O tipo de primitivas utilizado determina a estratégia de resolução de problemas empregado a cada passo. As estratégias podem ser, por exemplo, top-down, bottom-up, e inside-outside. As primitivas operam sobre entidades e relacionamentos, presentes no blackboard

a nível físico. Suas ações são vistas como eventos e fornecem base para o módulo de controle supervisionar a execução das fontes de conhecimento.

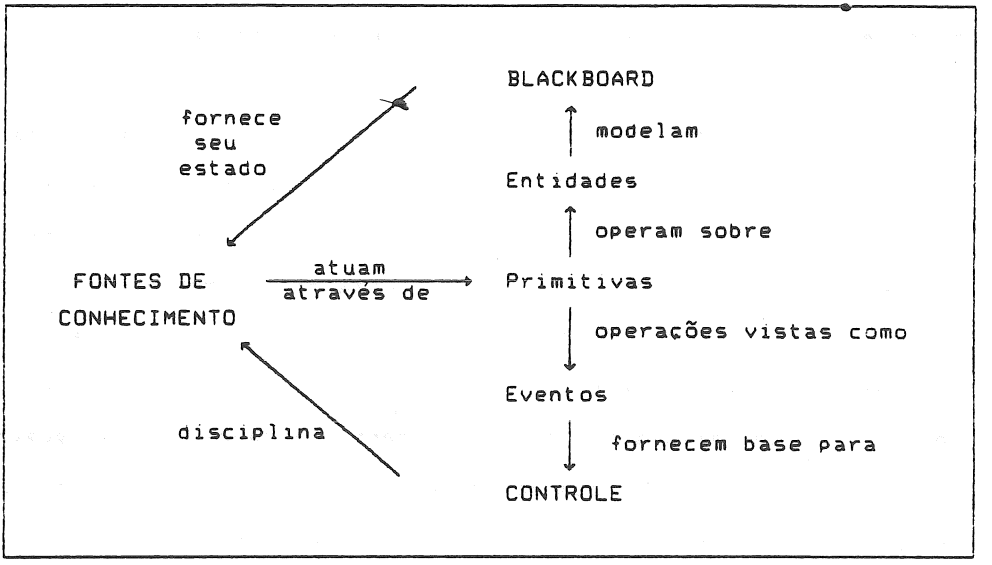


Figura 1 - Esquema Conceitual

3. O BLACKBOARD EM ASTUTO

Tanto flexibilidade quanto representação em diferentes níveis de abstração são facilmente obtidas através do Modelo Entidade-Relacionamento (MER), desenvolvido por Chen [01]. Desde a sua concepção inicial, vários pesquisadores vêm se preocupando em dar-lhe maior poder semântico, quer lhe acrescentando facilidades para especificação de restrições de integridade, quer lhe

possibilitando representar objetos em vários níveis de abstração. Como exemplo, temos o trabalho apresentado em [10], definindo facilidades para a representação de generalização, especialização e agregação de entidades, conduzindo a níveis diferentes de abstração. O conceito de "Transparência Parcial" proposto em [12] pode ser muito útil para a eliminação de redundâncias.

O Modelo Entidade-Relacionamento é bastante adequado para representar as entidades do blackboard, as relações entre elas, e também o nível de confiança das soluções parciais, expectativas de eventos e medidas de distância, normalmente manipuladas por Sistemas Blackboard. O MER foi utilizado inclusive para a representação física de todo o blackboard e para a construção da solução. A implementação do modelamento através da linguagem PROLOG foi também realizada com muita naturalidade, e os primeiros resultados têm encorajado muito a continuidade deste trabalho.

As entidades e os relacionamentos são representados por fatos Prolog. Embora as cláusulas apresentadas aqui possam ser entendidas sem conhecimento da linguagem, alguns leitores poder desejar consultar algum texto como [02].

Entidades para um restaurante e um clube poderiam ser:

```
estabelecimento(roma,restaurante,(rua_manaus,935),(11,22),40).
estabelecimento(regatas,clube,(av_parana,462),(8,20),90).
```

, onde "estabelecimento" seria o nome geral para todos os estabelecimentos; "restaurante" e "clube" representariam o tipo dos estabelecimentos; e em seguida teríamos seus endereços,

horários de funcionamento e expectativas de tempo de permanência, em minutos.

Abaixo, é dado um exemplo de relacionamento que mostra o trajeto entre o clube e o restaurante. Deixa-se o clube às 11:30, toma-se à avenida Paraná, segue-se pela rua Guanabara, e chega-se ao restaurante pela rua Manaus. O tempo de percurso é de dez minutos. A figura 2 ilustra este relacionamento.

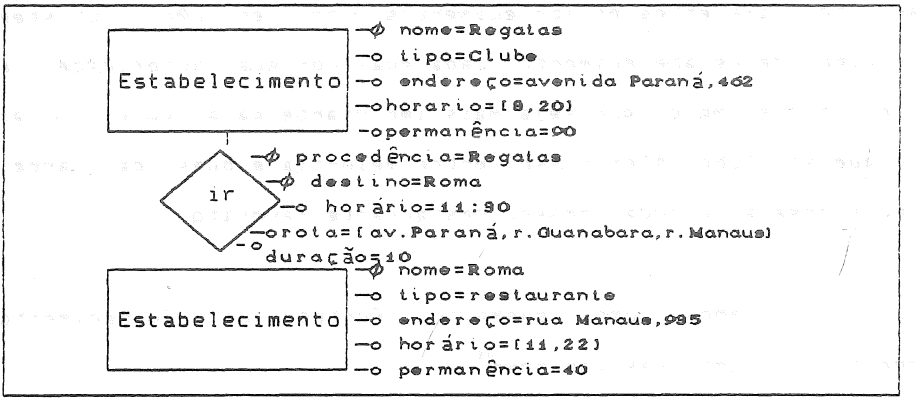


Figura 2 - Ilustração de Relacionamento

```
ir(regatas,roma,11.30,[av_parana,rua_guanabara,rua_manaus],10).
```

Os fatos (entidades e relacionamentos) presentes no blackboard são alterados pela ação das Fontes de Conhecimento, implementadas por regras Prolog.

4. OPERAÇÃO SOBRE O BLACKBOARD

Para solucionar o problema, as Fontes de Conhecimento agem modificando o blackboard. O procedimento das fontes é determinado por suas primitivas de ação, que por sua vez seguem uma estratégia de resolução de problemas.

Um exemplo é um sistema que determina uma rota em uma cidade, passando por diferentes estabelecimentos, como em [06]. Existem vários tipos de estabelecimento, cada qual com sua prioridade. É possível, por exemplo, que seja mais importante para você ir ao teatro que ao clube. Além disso, entre meio-dia e duas da tarde, você pode desejar almoçar em seu restaurante favorito.

Desta forma, temos como exemplo, Fontes de Conhecimento implementando primitivas top-down[1]:

a) que geram soluções alternativas.

Pode haver mais de um caminho de resolução que nos leve a uma solução satisfatória para o problema. Muitas vezes é desejável manter as soluções alternativas, como ilustrado na figura 3. No problema da rota pela cidade, verificamos que existem muitos trajetos entre dois estabelecimentos. Mudando o trajeto, pode-se conseguir visitar outros estabelecimentos próximos ao novo

(1) Nem todos os atributos são mostrados por motivos de clareza.

caminho. Dependendo da estratégia de resolução, pode ser desejável manter ambos os percursos.

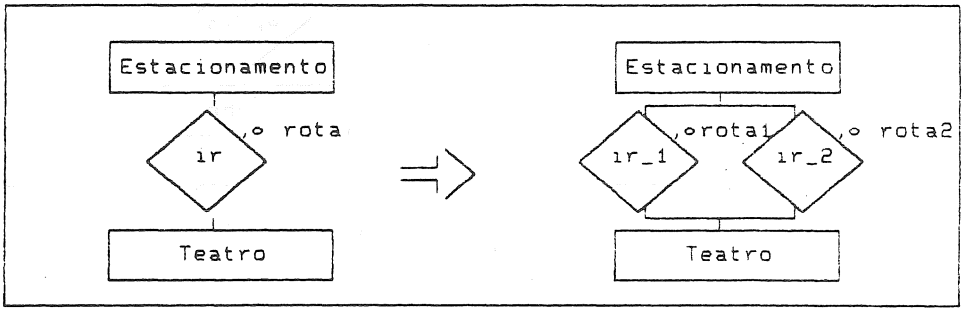


Figura 3 - Geração de soluções alternativas

b) que classificam as entidades.

Através da classificação de entidades, pode-se passar mais facilmente de um nível de abstração para outro, facilitando o raciocínio e a tomada de decisões (Figura 4). Classificar os estabelecimentos de acordo com sua localização pode, por exemplo, facilitar o trajeto na região. Isso viabiliza o planejamento das rotas por setores da cidade.

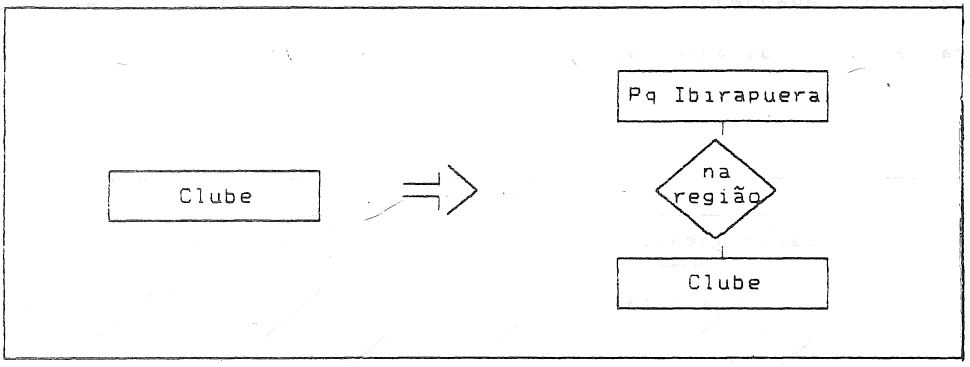


Figura 4 - Exemplo de Classificação

c) que dividem relacionamentos.

Podem existir entidades estreitamente relacionadas com outras. Muitas vezes, a representação dessa proximidade é desejável, e isto pode ser obtido através de divisão de relacionamentos (Figura 5). Na aplicação, pode ser proveitoso visitar certos estabelecimentos em sequência, por sua proximidade física, por exemplo. Além disso, se houver um relacionamento ir com horário=meio-dia, pode ser desejável ir a um restaurante antes de continuar o percurso.

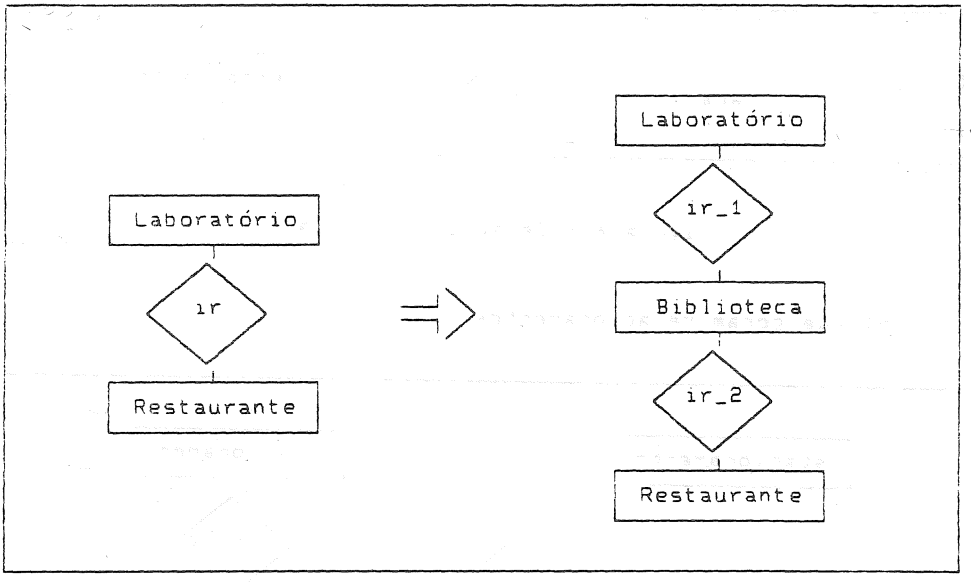


Figura 5 - Exemplo de Divisão de Relacionamento

Temos também as seguintes Fontes de Conhecimento implementando primitivas bottom-up:

a) que geram entidades.

Novas entidades passam a existir a partir da ocorrência de eventos externos, por exemplo. Primitivas de geração de entidades (Figura 6) inicializam sua existência no âmbito do sistema. Esse tipo de primitiva pode ser empregado, por exemplo, quando ao final do percurso houver tempo livre, inserindo-se um novo estabelecimento onde praticar esportes.

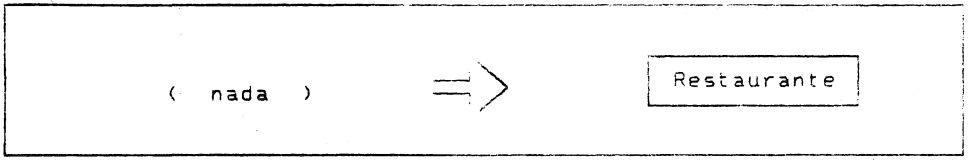


Figura 6 - Geração de entidade

b) que geram relacionamentos.

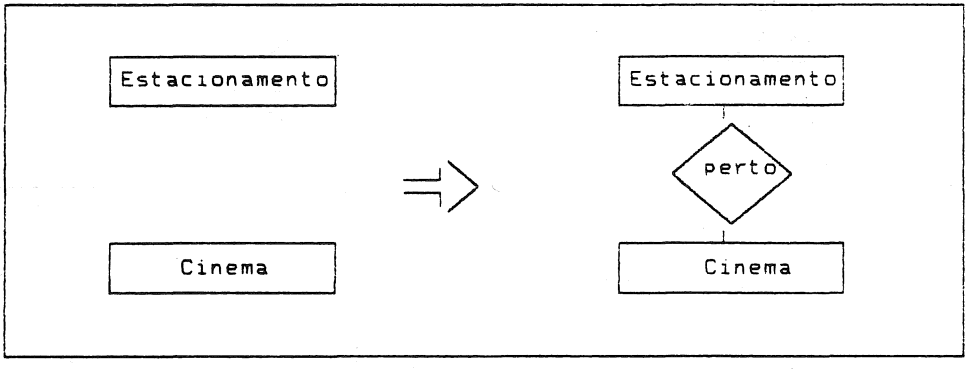


Figura 7 - Geração de relacionamentos

A geração de relacionamentos (Figura 7), além de contribuir para a construção da solução, é também uma maneira muito eficaz de melhorar o raciocínio das Fontes de Conhecimento, fornecendo a elas uma visão melhor do estado da solução.

c) que geram hierarquias.

A geração de hierarquias (Figura 8) permite que o sistema se abstraia de detalhes quando raciocina em níveis de abstração mais

altos. Este tipo de Fonte de Conhecimento pode também ser empregado de forma similar às de classificação, agrupando e privilegiando relacionamentos entre entidades afins.

Abaixo, temos um exemplo de classificação de entidades, que exemplifica como estas primitivas puderam ser implementadas:

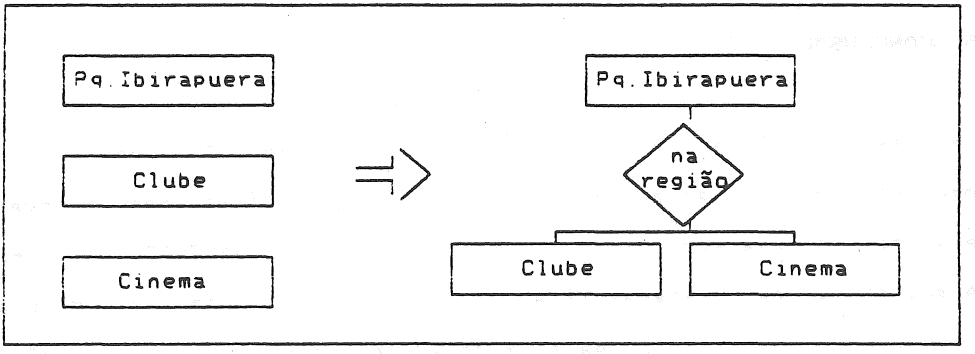


Figura 8 - Exemplo de geração de Hierarquia

```
ks_classificação_exemplo(região,Estabelecimento) :-
    na_região(Estabelecimento,Região),
    valida(Região),
    ins_blackboard(na_região,[Estabelecimento,Região]),
    ins_l_eventos(ger_classif,[na_região,Estabelecimento,Região]).
```

Nesse exemplo, ilustrado também pela figura 4, na_região é um predicado utilizado para determinar a região à qual o estabelecimento pertence; valida determina se a região determinada é de interesse e se o estabelecimento já não foi classificado. Os predicados ins_blackboard e ins_l_eventos introduzem o

relacionamento obtido no blackboard e na lista de eventos.

Todas as primitivas são tratadas como eventos pelo Módulo de Controle. Esse módulo simplesmente supervisiona a execução das Fontes de Conhecimento, auxiliando na escolha da estratégia mais apropriada.

5. CONCLUSÃO

Uma das características mais importantes do Modelo Blackboard é o oportunismo. Para alcançá-lo, as Fontes de Conhecimento devem enxergar o blackboard de forma apropriada. ASTUTO é uma Arquitetura de Sistemas Blackboard que utiliza o Modelo Entidade-Relacionamento para modelar e construir a solução, privilegiando a visão que as Fontes de Conhecimento têm do blackboard.

REFERÊNCIAS

[01] CHEN, P.P., The Entity-Relationship Model: Toward a Unified View of Data, ACM Transactions on Database Systems 01, (1976).

[02] CLOCKSIN, W.F. and MELLISH, C.S., Programming in Prolog, Springer-Verlag, 1984.

[03] CORKILL, D., GALLAGHER, K., andJOHNSON, P., Achieving

Flexibility, Efficiency, and Generality in Blackboard Architectures, in Proceedings of the Sixth National Conference on Artificial Intelligence, (1987), 18-23.

[04] ENGELMORE, R. and MORGAN, T., Blackboard Systems, Addison-Wesley Publ. Co., 1988.

[05] ENGELMORE, R.S., and MORGAN, A J., Introduction, in Blackboard Systems, Addison-Wesley Publ. Co., 1988.

[06] HAYES-ROTH, B., and HEWETT, M., BB1: An Implementation of the Blackboard Control Architecture, in Blackboard Systems, Addison-Wesley Publ. Co., 1988.

[07] NEWELL, A., Some Problems of the Basic Organization in Problem-Solving Programs, Proceedings of the Second Conference on Self-Organizing Systems, 1962, 393-323.

[08] PARNAS, D.L., On Criteria to be Used in Decomposing Systems into Modules, CACM 15, (1972), 1053-1058.

[09] PRESSMAN, R.S., Software Engineering - A Practitioner's Approach, McGraw-Hill Book Co., 1987.

[10] SMITH, J.M., and SMITH, D.C., Database Abstractions: Aggregation and Generalization, ACM Transactions on Database Systems 02, 1977.

[11] SofTech Inc., An Introduction to SADT, Manual, 1976.

[12] WAGNER, C.F., and MEDEIROS, C.B., Um Modelo Binário Estendido para Entidade-Relacionamento, Anais do XIV Seminário Integrado de Software e Hardware, 1987, 164-174.

CONNECTIONIST APPROACH FOR REPRESENTATION OF UNCERTAINTY IN KNOWLEDGE BASES

José Ramírez G., Irene Torres Hecker

PARADIGMA C.A.

Apartado 1087. Ciudad Guayana 8015

VENEZUELA

ABSTRACT

The intention of this work is to present an scheme for uncertainty representation in knowledge bases, based on two main aspects: the **relative importance** of evidences for reduction of uncertainty in conclusions and the **degree of belief** in each evidence, derived from its accuracy, perception method and subjectivism.

This work proposes a representation that expresses the relative importance of each evidence for the affirmance or negation of a conclusion, similar to the way it is done in a constraint satisfaction network. Also, the degree of belief -as we show- is naturally handled by this model.

In our proposal we consider a network with evidences as knowledge nodes formed by the following elements:

- A closed world (W)
- Defaults applicable (D)
- A threshold for the activation of such node (θ)

- A relative importance for the affirmance of a conclusion, similar to the weighted links of a connectionist model.(1)
- An inference engine that processes the nodes similar to a constraint satisfaction network.

THE CERTAINTY FACTORS

The use of certainty factors in the inference process on a Knowledge Base and their composition for a final conclusion are based on the criterion of increasing or decreasing their value according to the presence of certain evidences which contribute to affirm or deny a particular conclusion, inside a given alternative.

Though, if human behavior is to be modeled, there might be cases in which the presence or absence of evidences is not enough for someone to modify the previous certainty of a conclusion.

The certainty factors based upon probabilities will work adequately for cases which occur over a short period of time and considering a representative sample. For long term periods probabilities will change for statistical reasons and in consequence the certainty factors would be adjusted not necessarily reflecting reality.

There also will be present the problem of monotonic reasoning because the certainty factors already set won't be updated with the inclusion of new evidences in the knowledge base. Similarly, context for which certainty factors based on probabilities have been set, may vary too.

On the other hand, evidence is always obtained through a perception process which involves some degree of error, derived from the media used for its acquisition (human or devices). This is not represented when using certainty factors.

Additionally, the absence of evidences is not taken into account for the calculation of the certainty of the final conclusion, and eventually this could reduce the uncertainty related to it.

The initial setting of the certainty factors value is arbitrarily done and corresponds to the suggestion made by experts participating in the development of the Knowledge Based System. Nevertheless, after the System delivery these values may be subjected to an automated updating approach, and commonly these approaches are based on the decrease of the value of the certainty factors associated with seldom-firing rules, and the increase of the value of those certainty factors corresponding to rules which are frequently fired.

Very likely, the described process will lead to a limited Knowledge Base, unable to answer to the expected variety of situations originally posed. There will be "forgotten" rules, that in case of being applied won't reach a conclusion because of their "weakness".

EXPERIENCE, DEGREE OF BELIEF, IMPORTANCE

When evidences for a given conclusion are perceived by human beings, there is a "degree of belief" which represents the confidence they have in the accuracy of the values obtained. In fact, this feature won't necessarily change with the recurrence of the evidence over time.

The degree of belief on a given evidence -as we mentioned before- arises from the recognized subjectivism and perception error involved in the process of acquiring it.

In order to set a conclusion it is possible either to strengthen it with several evidences or to reduce the uncertainty by considering the absence of other evidences as a determining factor.

This defines relations between evidences and evidences, evidences and conclusions, conclusions and conclusions. This relations sustain the consistency of the Knowledge Base and add weight to the decision.

Each evidence, individually taken, has its own contribution to the affirmance of the conclusion which would be referred to as the "importance" of the evidence.

The degree of belief of an evidence and its importance are independent factors, and are independently managed in the model proposed. The former could be seen as the parameter which determines the influence field of a particular element in its value space, making it acceptable even when not exactly what expected. From this point of view it would be adjustable, but that would be done in a not dynamic way.

The degree of belief is conceptually a very stable value which will be updated just in case of detecting that it was wrongly set, or due to changes in the methods of perception.

Both, the degree of belief on the values, and the importance conceded to evidences are mainly based on experience.

THE CONNECTIONIST MODEL

As a representation for dealing in a natural way with the evidences, their degree of belief and importance relative to a conclusion we propose the use of a connectionist model, as shown in figure 1.

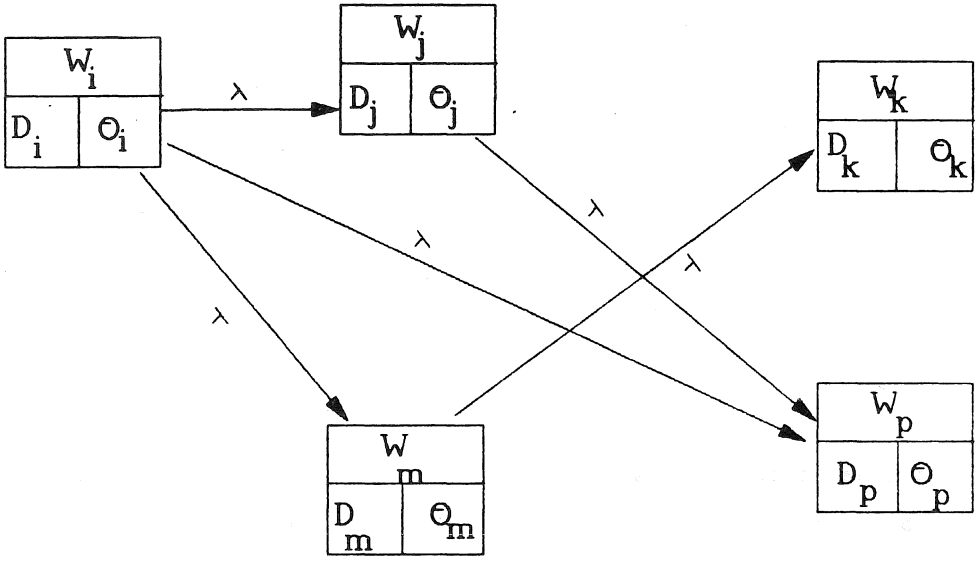


Fig.1 Connectionist model

each element of the network corresponds to a "Knowledge Node" (KN) formed by:

W

The closed World containing the rules related to the actual node (R), the facts (F) and the particular values (V).

D

The defaults applicable to determine the evidence in the node.

θ

The threshold for the activation of the knowledge node output.

Λ_{ij}

The links between Knowledge Nodes, corresponding to the value of the importance of each evidence to each other or to conclusions.

So, the Structure of the Knowledge Base (KB) would be as follows:

$$KB = \{(KN_1, \Lambda_1), \ldots, (KN_n, \Lambda_n)\}$$

$$KN_i = (W_i, D_i, \Theta_i)$$

$$W = \{(r_1, \ldots, r_n), (f_1, \ldots, f_n), (v_1, \ldots, v_n)\}$$

$$D = \{(d_1, \ldots, d_n), (e_1, \ldots, e_n)\}$$

$$\Lambda = (\lambda, KN_{target})$$

The degree of belief on each value for the evidences of a Knowledge Node is originally set for all those values obtained through perception. Every possible evidence will be represented in the model. Taking advantage of the network structure, the correlation between the individual evidences will naturally occur.

THE INFERENCE

The inference process on the described model will take place similar to a constraint satisfaction network. The goal is to obtain the conclusion for which the greater number of more

important evidences are established. The inference process consists of a set of non-probabilistic functions, as will be shown.

Considering three nodes: i,j,p, the inference engine process will determine the following values:

E_j

the evidence in the actual node, the value of a composition function of the evidences, defaults of the world and the world itself, defined as:

$$E_j = k(Net_{ij}, D_j, W_j)$$

where Net is the vector of incoming evidence, defined as:

$$Net_{jp} = (E_j, S_{jp})$$

where S_{jp}

is the strength of the connection between nodes j and p defined as:

$$S_{jp} = h(A_j, \lambda_{jp})$$

where λ_{jp}
is the importance of ...,
as evidence for p and A_j
is the activation function defined as

$$A_j = f(S_j, \theta_j)$$

where θ_j
is the threshold in the node and S_j
the strength of the evidence, is a function calculated on the degree of belief of the different values of the evidence and the strength of the connection thru which they came to the actual node. It is defined as:

$$S_j = g\left((\beta_1, \ldots, \beta_n)_j, S_{ij}\right)$$

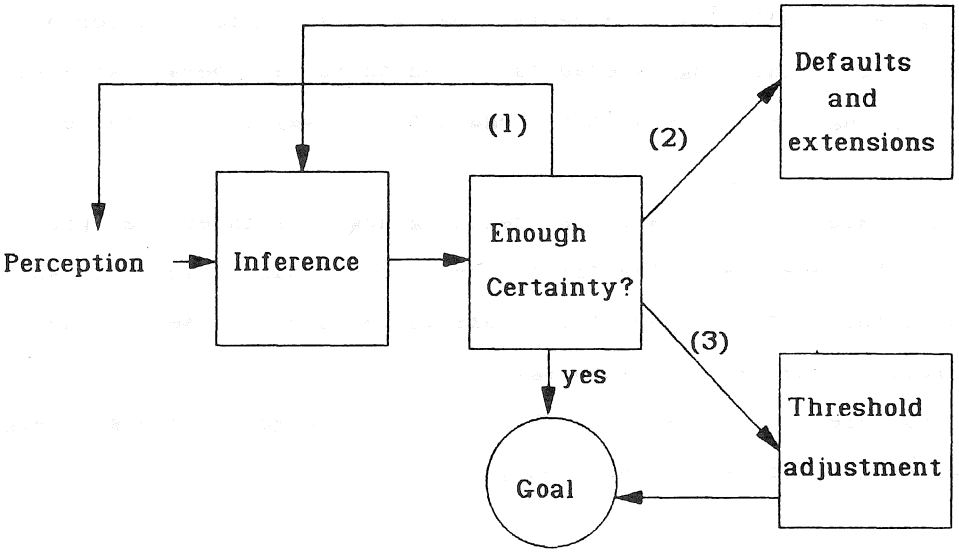
Where β_i
is the degree of belief of value i

In the connectionist models, particularly in the Ndimensional Coulomb Model, there are typically three strategies for conflict resolution: A conservative one in which a result will be accepted only if it has been set with a unique value whose certainty is greater than a given threshold. A moderate approach where some

degree of ambiguity is accepted, being possible to look for other sources of knowledge (Defaults); and third, a liberal strategy, accepting a solution with no regard to the way it is obtained.

There are two main aspects defining how the inference process will be take place. The first one is the establishment of where will the conflict resolution take place, and the second one is properly the selected strategy for each conflict resolution point considered in the network. The inference process on the knowledge nodes not defined as decision points will be parallel.

According to what we just mentioned the inference process would be as shown in figure 2.



(1) Conservative approach
(2) Moderate approach
(3) liberal approach

Fig. 2 Inference process

The results of the calculation of the evidences would be a conclusion with its associated certainty. If the value of the certainty surpasses the value of acceptance -even in a conservative mode- the conclusion will be valid. But, if the value of the certainty is under the acceptance value there are several possible actions depending on the desired answer, at a decision point.

From a conservative point of view (1), a new data acquisition -perception- and total redoing of the inference process would be necessary.

In a moderate approach (2), new sources for completing the knowledge involved in the inference would be used. Specifically we propose the conduction of Commonsense Reasoning based on Defaults Theory. Commonsense is made up of positive evidence and absence of contradictory evidence. Normally there are more negative facts to be said about a particular conclusion - things against other conclusions, than things to positively affirm it. So, negative facts do are relevant information in a decision-making process. On the terms we just described, Defaults Theory seems to fit really well for the affirmance of facts when there is no reason for denying them, because of the consideration of the possible presence of contradictory facts.

A default value will be calculated and asserted for each evidence which couldn't be set during inference.

Finally, when working in liberal mode (3), any conclusion at which the process could have arrived would be taken neglecting its certainty value.

So, there will always be present -in one way or another- the

incidence of the missing evidences.

ADAPTATION AND LEARNING

The adaptation of the Knowledge Base is expected to be accomplished through different mechanisms. They will depend on the contribution of each inference process executed on it, which varies upon the strategy selected for conflict resolution in the decision points of the network.

In the case of moderate and liberal strategies, the defaults are used and proper extensions are generated as explained in (Eckmiller 1.987), these extensions are valid only in the session or context in which they are being used. The generation of extensions gives an adaptive behavior to the model. As mentioned before, in the liberal strategy thresholds are just reference values.

For an specific inference process, if a moderate or liberal conflict resolution strategy is to be used at any point, recurring to the revision of defaults and generation of extensions then the importance of the particular evidence being considered should be lessen to reflect that it wasn't constructed as rigorously as possible when using the determined values.

The training of the model follows a multi-layer network approach; considering case studies presented to the model and confronting the output with the expected solution, calculating the error and propagating it back through the net, causing the adjustment of the thresholds and the importance factors. The cases are initially defined by the knowledge engineer and are updated with the unsolved situations presented to the model and the solutions given by experts.

In every decision point where a conservative conflict resolution approach will take place it would be necessary to have -talking in connectionist jargon- a supervised scheme of learning in which the system could be told about the adequacy or inadequacy of its answer.

If a particular answer was expected and not reached then the evidences participating in the inference should decrease their threshold value. But an evidence taken with less elements shouldn't be as important for the conclusion as if it were taken with all its elements determined through reliable sources.

If a particular conclusion was reached and the evidences participating in the process are consciously not considered enough to support that conclusion (there is no adequate correspondence between the expected and the real answer of the

system), then the threshold values of all these evidences should be increased, and their importance may be decreased meaning that it's necessary the presence of other evidences to reach the conclusion.

CONCLUSIONS

1. The use of a connectionist model for structuring a Knowledge Base composed by Knowledge Nodes allows to represent elements such as evidences, their associated degree of belief and their importance relative to other evidences or conclusions. These parameters seem to offer a more reliable way for managing uncertainty than previously known approaches.

2. The inference process similar to that of a constraint satisfaction network and conducted based on the conflict resolution strategy adopted for the relevant decisions, allows to reduce the uncertainty inherent to conclusions of the System, by taking into account the missing evidences and the combined action of multiple evidences.

3. The proposed representation and inference method guarantee an updating mechanism based on the inclusion of new evidences through extensions to the Knowledge World.

4. Future work will be oriented toward an implementation in an object oriented environment. Also, more detailed treatment of the aspects related to learning and adaptation will be studied.

REFERENCES

Anderson, J. and Rosenfeld, E. (Editors); "Neurocomputing: Foundations of Research", MIT Press, 1988.

Biermann, A.W.; "Fundamental Mechanisms in Machine Learning", Advanced Topics in Artificial Intelligence, pp 125-145, Springer Verlag, NY, 1987.

Buchanan, B. and Shortlife, E.; "Rule-based Expert Systems", Addison-Wesley, 1984.

Cohen, P.; "The Control of Reasoning under Uncertainty: A Discussion of Some Programs". Department of Computer and Information Science. University of Massachusetts. Amherst.

Collins, A. and Smith, E. (Editors); "Readings in Cognitive Science", Morgan Kaufmann Publishers, 1988.

Deng, P.; "A Skill Refinement Learning Model for Rule-Based Expert Systems", IEEE EXPERT, April 1990.

Dunne, P.; "The Complexity of Boolean Networks", Academic Press, 1988.

Eckmiller, R. and Malsburg, Ch. (Editors); "Neural Computers", Springer Verlag, 1987.

Etherington, D.; "Reasoning with Incomplete Information", Morgan Kaufmann, 1988.

Grossberg, S. (Editor); "Neural Networks and Natural Intelligence", MIT Press, 1988.

Lamberti, D. and Wallace, W.; "Presenting Uncertainty in Expert Systems: An Issue in Information Portrayal". Technical Report No. 37-87-134. Rensselaer Polytechnic Institute. September 1987.

NG, K.; "Uncertainty Management in Expert Systems", IEEE EXPERT, 1990.

Ramirez, J.; "Neural Networks: A Tutorial". Third National Conference on Artificial Intelligence. Venezuela 1990

Rumelhart, D. and McClelland, J.; "Parallel Distributed Processing". Vol. I,II and III, MIT Press, 1986.

Zadeh, L. A.; "Commonsense and Fuzzy Logic", The Knowledge Frontier, pp 103-136, Springer Verlag, NY, 1987.