PANEL '86
EXPODATA

XII CONFERENCIA LATINOAMERICANA DE INFORMATICA

3 AL 7 DE NOVIEMBRE 1986
MONTEVIDEO - URUGUAY.

XII CONFERENCIA LATINOAMERICANA DE INFORMATICA

La XII Conferencia Latinoamericana de Informática persigue, al igual que las anteriores, el establecimiento de una creciente profundización académica y de fuertes vínculos entre la comunidad informática latinoamericana que favorezcan el desarrollo integrado de esta disciplina en la región.

El carácter itinerante de estas conferencias tiene como objetivo convertirlas en un factor de promoción e incentivación de las actividades científicas y de investigación en los países en que se desarrollan.

3 AL 7 DE NOVIEMBRE 1986
MONTEVIDEO - URUGUAY.

COMITE DE HONOR

Ing. LUIS ABETE
Decano de la Facultad de Ingeniería
Uruguay

Dr. CARLOS CORREA
Sub-Secretario de Informática y Desarrollo
Argentina

Dr. JOSE R. DORIA PORTO
Secretario Especial de Informática - Brasil

Dr. JORGE LUIS ELIZALDE
Intendente de Montevideo - Uruguay

Ing. VICTOR GANON
Asesor de Informática Presidencia de la
República - Uruguay

Ing. JUAN GROMPONE
Prof. del Instituto de Computación de la
Fac. de Ingeniería - Uruguay

Cr. ENRIQUE IGLESIAS
Canciller de la República - Uruguay

Senador Dr. LUIS ALBERTO LACALLE
Pte. de la Comisión de Informática y
Prospectiva del Senado - Uruguay

Ing. FERNANDO LAZCANO
Pte. de la Soc. Uruguaya de Informática
Uruguay

Dr. SAMUEL LICHTENZJTEIN
Rector de la Universidad de la República
Uruguay

Dr. GUSTAVO MALEK
Director de la Oficina Regional de
UNESCO - Uruguay

Sra. MONICA MASSEY DE HOYOS
Representante Residente Adjunto de
Naciones Unidas - Uruguay

Sr. ARTHUR PEREIRA NUNES
Secretario Ejecutivo de ABICOMP - Brasil

Ing. Agr. CARLOS PEREZ ARRARTE
Secretario Ejecutivo de CIEDUR - Uruguay

Ing. FELIX A. PIMENTEL
Presidente de la AUDEPI - Uruguay

Dra. ADELA RETA
Ministro de Educación y Cultura - Uruguay

Dr. MANUEL SADOSKY
Secretario de Ciencia y Técnica - Argentina

Cr. ISAAC UMANSKY
Presidente de la Comisión Nacional de
Informática - Uruguay

Dr. JORGE VIDART
Presidente del CLEI - Uruguay

COMISION ORGANIZADORA

ANA ASUAGA
Ministerio de Educación y Cultura

JUAN BOLOGNINI
Sociedad Uruguaya de Informática

INES CAMOU
Intendencia M. de Montevideo

GUZMAN ETHEBEHERE
Asoc. Urug. de Prof. en Informática

ALBERTO GONZALEZ
CIEDUR

OMAR PAGANINI
Asoc. de Ingenieros del Uruguay

ALVARO TASISTRO
Inst. de Computación de la Universidad de
la República

Coordinador General
DANIEL E. GASCUE
Centro Latinoamericano de Estudios
en Informática

Secretaria Ejecutiva
RAQUEL BARREIRA

COMITE DE PROGRAMA

Presidente
JUAN J. CABEZAS
Univ. de la República - Uruguay

SERGIO FLORES
Escuela Politécnica del Litoral - Ecuador

ROBERTO MANOEL MACEDO
Universidad Federal de Río Grande do Sul
Brasil

PATRICK O'CALLAGHAN
Universidad Simón Bolívar - Venezuela

JOSE PINO
Universidad de Chile - Chile

JORGE PLUSS
Universidad Nacional de Rosario -
Argentina

VICTOR MANUEL TORO
Universidad de los Andes - Colombia

MAGIN ZUBIETA VILLEGAS
Universidad Mayor de San Andrés - Bolivia

CONSEJO DIRECTIVO CLEI

Presidente
JORGE VIDART
ESLAI - ARGENTINA

Secretario Ejecutivo
ALDO MIGLIARO OSORIO
Univ. Católica de Valparaíso - CHILE

DIEGO ANDRADE STACEY
Univ. Católica de Ecuador - ECUADOR

NEWTON BRAGA ROSA
Univ. Federal de Río Grande do Sul -
BRASIL

CARLA BRUSCHI DE CARDINALE
Univ. Nacional de San Juan -
ARGENTINA

CARLOS CIUFFARDI PACE
Univ. Católica de Valparaíso - CHILE

DANIEL E. GASCUE
Centro Interdisciplinario de Estudios sobre
el Desarrollo, Uruguay (CIEDUR)
URUGUAY

GERMAN HERNAN
Asoc. Colombiana de Cálculo Electrónico
e Investigación Operativa - COLOMBIA

WILFREDO KLEEBERG
Asoc. Peruana de Computación e
Informática - PERU

LUIS A. MEYER
Univ. Católica de Asunción - PARAGUAY

CORRADO VALLET
Univ. Gabriel René Moreno - BOLIVIA

ORGANIZACION

CLEI
UNIVERSIDAD DE LA REPUBLICA
CIEDUR
AUDEPI
MINISTERIO DE EDUCACION Y CULTURA (URUGUAY)
SOCIEDAD URUGUAYA DE INFORMATICA
ASOCIACION DE INGENIEROS DEL URUGUAY
INTENDENCIA MUNICIPAL DE MONTEVIDEO

AUSPICIO

UNESCO
CREI
PROTEC-OEA/MEC/UFRGS

PATROCINIO

ARNALDO C. CASTRO S.A.
BULL DEL URUGUAY S.A.

COLABORACION

BURROUGHS DEL URUGUAY
COASIN URUGUAYA SRL.
COMSYS S.A.
IBM DEL URUGUAY S.A.
INTERFASE LTDA.
MICROCOM LTDA.
NCR DEL URUGUAY
OLIVETTI URUGUAYA S.A.
PLUNA
QUANTUM CONSULTORES ASOCIADOS
SISTEMAS CADE
SYCOM LTDA.
XEROX URUGUAY S.A.

PRESENTACION DE LOS TRABAJOS

Los trabajos expuestos a continuación fueron agrupados de acuerdo a cinco grandes líneas temáticas. Si bien los temas tratados por los trabajos de cada grupo son bastante diversos y resulta difícil asignar un título que los represente a todos, los trabajos del grupo 1 están relacionados con la Informática Distribuida y Bases de Datos, los del grupo 2 con Complejidad, análisis de algoritmos y análisis de eficiencia, los del grupo 3 con Proyectos y aplicaciones informáticas; las del grupo 4 con los Lenguajes de Programación y las del grupo 5 con Informática y Sociedad.

En las páginas finales se incluyen algunas comunicaciones cortas, cuyo contenido fue analizado por el Comité de Programa solamente en cuanto a su pertinencia temática.

GRUPO I

PRACTICAL ISSUES ON CONCURRENCY CONTROL IN DATABASE SYSTEMS

M. R. S. Borges
Universidade Federal do Rio de Janeiro
Rio de Janeiro - *Brasil*

I - INTRODUCTION

For the last few years the area of concurrency control in database systems has been the object of a lot of research [ROSE78, BERN81, KOHL81, GARD82]. Until mid 70's it was generally accepted that locking policies were the best way to solve conflicts in a multi-processing environment. However, with the rise of distributed systems, the research for alternative approaches started again. The reason for this was the fact that communication costs play an important role on the overhead of process interaction. Also, because it is important to keep the site autonomy as the maximum possible. Several other alternatives have been proposed and many more will appear as was explained in [BERN82].

This paper does not introduce a new mechanism for concurrency control. Neither does it compare performance of different strategies. While these research works are considered quite important, we thought that would be interesting to throw some light on the practical aspects of the subject. By practical we mean those of benefit to real applications. This paper tries to demonstrate how the practical aspects relate to the research being carried out on the problem.

It is our opinion that if some factors are taken into consideration, it can improve the performance and the usability of concurrency control mechanisms for database software.

We examine three issues of database usage as it relates to:

- the notion of conflict;
- the types of application;
- integrated design.

We show in each of the following sections how these three issues may affect the design decisions of a concurrency control mechanism.

II- THE NOTION OF CONFLICT

Any set of transactions that access concurrently the same object in an incompatible mode (i.e. at least one is an update) is said to conflict {ESWA76}. The usual way to avoid conflicts is to serailize the conflicting access, in the same order in all objects. An attempt to override this rule would cause one of the following effects:

- an inconsistent update;
- a lost update;
- obsolete versions apparent to the user;
- inconsistencies apparent to the user.

The first two effects occur when two update transactions are performed on the same object concurrently. The inconsistent update is the result of overlapping actions of update transactions on objects related by some integrity constraint. The lost update is a result of an update based on obsolete (but consistent) data. In other words inconsistencies are caused by inappropriate Reads within update transactions. The lost update is caused by either inappropriated Read or Write actions within update transactions. The reader is referred to {BORG86} for more details and examples.

The last two effects may occur when processing Read and Update transactions in parallel. They do not constitute a problem for the database itself but for the user accessing it. This is true as long as the result of a read would not be used for further update, i.e. the Read action is part of a read-only transaction. The third effect occurs when the reads are issued on partially updated objects related by some integrity constraint. The fourth is caused by anticipating a read access on an object with some previous update transaction pending on it.

Whichever notion of conflict is followed, a mechanism is said to be correct if it prevents conflicts happening. In the case of update conflicts (first two effects), we agree that there is no sense in relaxing the conditions which cause them. However, we believe that for the majority of applications is does not constitute a problem if the user sees an obsolete database (how obsolete it can be is dependent on the application). We see three main reasons for that.

First, it is impossible for most applications to keep the databse in compass with the real world. it maybe the case that some facts are waiting to be stored in the database; then the version available is an obsolete one. Also there is no guarantee that the version the transaction was based on would be still valid when the user receives it, unless he prevents real world change occurring. It is our opinion that because of the nature of some applications, the effort the concurrency control mecanism makes to avoid obsolete versions for retrieval-only transactions is fruitless.

Second, other components of the system may arbitrarily contribute to this obsolence. The network, for example, does not usually assure that the messages will be received in the order they were transmitted. In that case the order of queries would be changed inside the system and this can make the answers obsolete.

Last but not least, the system can benefit a lot from permiting more paralellism. Simulation studies on protocols which relax this condition show that the performance of the concurrency mechanism can improve substantially in the case of "query predominant" databases [BORG85, BORG86].

These three resons show that obsolete versions are common-place in several applications. The users are naturally aware to this. The concurrency control mechanism should not bother in avoiding them unless the users express they want an "up-to-date" version.

Actually, these arguments are proven not to be completely true for some mechanisms. The best example is the multiversion protocol [BERN83], which use obsolete version in case of a transaction arrives after a later update has been processed in the same object. The serializability theory only guarantees that the execution will be equivalent to some serial one [BERN82]. This means that these protocols may permit a Read transaction to proceed even if a Write action, from a transaction which arrived earlier, is to be performed later on the same object.

However, most of the mechanism do not permit Reads while Writes are in progress, even if a two-phase commit protocol has been used. In the basic time-stamping ordering approach an out-of-order Read can cause the abortion of update transaction [BERN80]. In the optimistic approach, the validation of a Read transaction may cause the invalidation of an Update transaction later [KUNG81]. In a multiversion protocol an Update transaction can be invalidated by a Read action because the protocol does not diferentiate between Reads for update and Reads for retrieval-only transactions [BERN83].

The fourth effect is a more serious one. Relaxing the conditions which produce it would lead to more parallelism, but the result does not seem acceptable for most of applications. The only value of those results may be for statistical or uncompromised queries [BORG85].

It is our opinion that any concurrency control mechanism should provide the user with the option of relaxing the conditions which produce these last two effects. By doing this it can achieve better system performance when these effects do not matter, as estabilished by the application requirements. The concurrency control mechanism should be able to follow different protocols according to the user definition of transaction. The transactions may be divided into two classes; the normal and the

weak class of transactions. Some recent proposals modifying the basic appraches show that research is starting to be oriented on this direction (UNLA83, GARC83).

III - TYPES OF APLICATIONS

It has been the conclusion of several papers which compare concurrency control mechanism that the performance is very dependent, apart from other things, on the type of application (BADA80, LIN82, LIN83, KOHL83, PEIN83). There is general recognition that any mechanism can outperform another if we select the appropriate aplication. Because of the characteristics of the concurrency problem, it seems unlikely that any future approach can claim that it has the best performance in all situations.

Now, these mechanism are supposed to be implemented in database softwares for general use. How should the user (DBA) react if he knows that the kind of aplication he has is not of the type in which the mechanism produces the best results? Should he have to choose the package according?

We believe the answer to this question is that the software has to allow more flexibility on the concurrency control mechanisms. This can be achieved by providing the software with a selection of concurrency control strategies similar, for example, to the choice it provides for access methods. The concurrency control should be seen as a interchangeable component in which the user (DBA) may have the option of selecting the most appropriate strategy.

Another alternative, perhaps more feasible, is to design a mechanism which can make the decision on the concurrency control protocol dynamically, according to the kind of transactions running. An example is the variable granularity locking protocol (KORT81). In that approach, the size (granule) of the object referenced by the transaction.

We think that some research effort should be directed to design a more general mechanism which can be sensitive to the kind of transactions running on the system. At the moment it seems clear that those mechanism should incorporate ideas from various approaches in order to achieve great efficiency in all cases.

IV - INTEGRATED DESIGN

Most papers about concurrency control tend to analyse the problem as if it were the central issue in physical design in database systems. All the other parts of the system are seen only as they affect the concurrency mechanism. While this approach is useful for studying the algorithms it may cause several problems when we try to integrate the mechanism with the other components of the system.

The efficiency of the concurrency control is usually measured by the throughput produced from the input load. It has been demonstrated somewhere that this is very dependent on the characteristics of the load. Furthermore, we claim that the performance of a concurrency control mechanism measured by its throughput, does not necessarily mean the best overall performance of the system.

That aspect is being ignored by researchers when comparing different mechanisms. The throughput produced by a concurrency control mechanism is a potential one because it depends on parallelism in other parts of the system (disks, lines, etc). As the performance of a mechanism is very sensitive to parallel execution, we can have the **"do-all-for-nothing effect"** {BORG85}. This occurs when the mechanism spends a lot of time is discovering a non-conflictant parallel execution, yet at the end the actions would be serialized because of other constraints.

In our opinion, the search for better concurrency control protocols must be oriented to one which produces the best overall performance of the system. In saying this, we mean that concurrency control implementation must be sensitive to the particularities of the other components of the database and not viewed as a stand-alone feature.

Another aspect, perhaps not so visible but also very important, is to investigate how the concurrency mechanism affects other components of the system. The best example is the recovery protocol. Some concurrency mechanism are so complex that would turn it almost impossible to recover from a crash without implementing an algorithm as complex as the concurrency mechanism itself. It is assumed that a history ought to be reproducible when we have a crash. In other words a recovery from a crash must not alter the effect of transactions already commited. However, in some cases, in order to reproduce such history the recovery mechanism may have even to reproduce abortions (???). More important, the recovery can be very expensive and cause long delays in the system.

V - CONCLUSIONS

We have presented in this paper some ideas of how to deal with concurrency control in a more realistic way. While it has been important to search for new algorithms we think some effort should now be directed to producing mechanisms for use in real systems and not in hypothetical ones. We believe that at the moment there is a gap between the research and the implementation that has not been properly filled.

There are other aspects to be taken into consideration which were not covered in this paper. One of them is the uniformity of processing. There is little sense, for real applications, in arbitrarily speeding some transactions if that causes unacceptable delays on other transactions. (for example, due to sucessive fails).

All these ideas has been investigated in a Ph.D. project [BORG86] which simulates the performance of the mechanisms in practical situations. We incorporated in the simulation model different kind of transactions which eases the notion of conflict. We are experimenting with a wide range of application types (from query to update predominant system). We modeled all the components of a database at the physical level in order to examine the effects the concurrency mechanism has on other components as well.

REFERENCES

[BADA80] Badal, D.Z. : The analysis of the effects of concurrency control on distributed databases system performmance. 6th VLDB (1980) pp. 376-383..

[BERN80] Bernstein, P. and Goodman, N. : Timestamp-based algorithms for concurrency control in distributed databases, 6th VLDB (1980) pp. 275-284.

[BERN81] Bernstein, P. and Goodman, N. : Concurrency control in distributed databases systems. ACM Computing Surveys V13 N2 (June 1981) pp. 185-221.

[BERN82] Bernstein, P. and Goodman, N. : A sophisticate's introduction to distributed database concurrency control. 8th VLDB (1982) pp. 62-76.

{BERN83} Bernstein, P. and Goodman, N. : Multiversion concurrency control - Theory and Algorithms. ACM TODS V8 N4 (December 1983) pp. 465-483.

{BORG85} Borges, M.R. : Twords a flexible mechanism for concurrency control in database systems. 4th British National Conference on Databases. Keele, UK (1985) pp. 39-60.

{BORG86} Borges, M.R. : A flexible mechanism for concurrency control in systems. Ph.D. Thesis 1986, School of Information Systems, University of East Anglia, UK.

{ESWA76} Eswaran, K.P., Gray, J. Lorie, R.A. and Traiger, I. : The notions of consistency and predicate locks in a database system. Comm. ACM V19 N11 (November 1976) pp. 624-633.

{GARC83} Garcia-Molina, H. : Using semantic knowledge for transaction processing in a distributed database. ACM TODS V8 N2 (June 1983) pp. 186-213.

{GARD82} Gardarin, G. and Melkanoff, M. : Concurrency control principles in distributed and centralized databases. INRIA Rapport de Recherche no. 113 (January 1982).

{KOHL81} Kohler, W.H. : A survey of techniques for synchronization and recovery in decentralized systems. ACM Computing Surveys V13 N2 (June 1981) pp. 149-183.

{KOHL83} Kohler, W.H., Wilner, K.C. and Stankovic J.A. : An experimental comparison of locking policies in a testbed database system. ACM SIGMOD Conference (1983) pp. 108-119.

{KORT81} Korth, H.F. : A deadlock-free, variable gralularity looking protocol. Proc. 5th Berkeley Workshop on Dist. Data Manag. and Computer Networks. (February 1981) pp. 105-121.

{KUNG81} Kung, H.T. and Robinson J.T. : On optimistic methods for concurrency control. ACM TODS V6 N2 (June 1981) pp. 213-226.

{LIN 82} Lin, W.K. and Nolte, J. : Performance of two-phase looking. Proc. 6th Berkeley Workshop on Dist. Data Manag. and Comp. Network. (February 1982) pp. 131-160.

{LIN 83} Lin, W.K. and Nolte, J. : Basic timestamp, multiple version, and two-phase looking. 9th VLDB (1983) pp. 109-119.

[PEIN83] Peinl, P. and Reuter, A. : Empirical comparision of database concurrency control shemes. 9th VLDB (1983) pp. 97-108.

[ROSE78] Rosenkrantz, D.J., Stearns, R.E. and Lewis, P.M. : System level concurrency control for distributed database systems. ACM TODS V3 N2 (June 1978) pp. 178-198.

[UNLA83] Unlad, R., Praedel, U, and Schlageter, G.: Design alternatives for optimistic concurrency control schemes. 2nd ICOD (International Conf. on Databases). (August 1983). pp. 288-297.

BANCOS DE DATOS GEOGRAFICOS Y MODELO RELACIONAL

Hector Daniel Castro
Porto Alegre - *Brasil*

INTRODUCCION

Entre las aplicaciones no convencionales de los bancos de datos se encuentran aquellas dedicadas a los esfuerzos de planificación, en las que el acúmulo de datos organizados geográficamente juega un papel importante. Si bien estos datos pueden ser de tipos muy diversos: climáticos, hídricos, económicos, demógraficos, etc., su característica común es que de alguna manera están relacionados con una ubicación geográfica. Las organizaciones propuestas para los bancos de datos geográficos utilizan en su mayoría el modelo de datos relacional. Se analizarán las maneras de representar los datos geográficos y tres implementaciones de bancos de datos geográficos. Esta presentación forma parte de un proyecto orientado a obtener un esquema conceptual de un banco de datos geográficos.

REPRESENTACIONES DE DATOS GEOGRAFICOS [1]

Conforme fue explicado anteriormente, en un banco de datos geográficos existen, en principio, dos tipos de datos: aquellos que representan propiedades específicas del tema siendo estudiado (por ejemplo, número de personas o valores económicos), que serán denominados "datos simbólicos"; y aquellos que representan la ubicación espacial (geográfica) a que están referenciados los primeros. Estos ultimos serán llamados "datos geográficos". Es claro que entre los datos simbólicos pueden existir sub-clasificaciones, o incluso pueden no aparecer por completo, dependiendo de la aplicación. En cambio, los datos geográficos deben estar siempre presentes, y para ellos debe usarse en todo el banco de datos una representación uniforme.

Para objetivar, vamos a dar un ejemplo. Supongamos un banco de datos destinado a describir la red vial de una región. Si consideramos que dicha red esta compuesta por rutas que unen ciudades y nuestro interés fuera hallar la manera de ir de una ciudad a otra, bastaría con representar la ciudad de origen y la ciudad de destino de cada ruta, y tal vez las intersecciones de las rutas. Aquí sólo existe información geográfica, si bien que de manera implícita a través de la ubicación espacial de cada ciudad, que suponemos conocida, y que, obviamente, no varía con el tiempo. Si, además, estuviéramos interesados en hallar la mejor ruta, basándonos en criterios como tipo de la calzada o volumen de tránsito, deberíamos incluir esos datos en

nuestro banco. Estos últimos datos pertenecen a los que llamamos simbólicos. Los datos simbólicos pueden ser de dos tipos: numéricos (volumen de tránsito) o identificatorios (tipo de calzada). Vemos que todos los datos están relacionados espacialmente, aunque sea en forma indirecta: el tipo de calzada se refiere a una ruta, y ésta es ubicada por sus puntos extremos.

Cabe aclarar que, en nuestro concepto, un banco de datos como el descripto no sería propiamente un banco de datos geográficos. Para ser tal, falta un concepto importante, que es el de medida. En el banco de datos presentado, no podemos averiguar la cantidad de kilómetros que una ruta emplea para unir dos ciudades. Para conseguir eso deberíamos agregar explícitamente la ubicación de cada ciudad, mediante sus coordenadas geográficas, si suponemos que las rutas son rectas. Si no son rectas, deberíamos descomponerlas en segmentos rectos. Nótese que no bastaría con agregar esa información a los datos simbólicos, porque queremos asimilar el uso del banco de datos geográficos al uso de un mapa. Es decir, deberían poder realizarse medidas encima del banco de datos de la misma manera como son hechas encima del mapa.

Existen dos maneras usadas comunmente para representar datos geográficos, que permiten realizar mediciones. El primer método, que llamaremos "de reja", consiste en cuadricular el área cubierta por los datos, y ubicar espacialmente cada elemento de área por el número de fila y el número de columna, y las coordenadas de una de las esquinas de la cuadrícula. Los datos simbólicos son entonces referidos a estos elementos de área. En el caso de variables del tipo $z = f(x,y)$, asociaremos con cada elemento de área un número o un identificador. Este es el tipo de datos para el que es más útil el método de reja. Si queremos representar datos lineales, como por ejemplo una vía de ferrocarril, deberíamos recurrir al expediente de indicar su presencia en cada uno de los elementos de área que ella atraviesa, y su ausencia (por "default") en todos los otros. Es de hacer notar que usando este método es fácil calcular la superfície relacionada con un atributo, simplemente multiplicando la superfície de un elemento de área por la cantidad de elementos con ese atributo. Igualmente fácil es calcular la distancia entre dos elementos de área. Una variante de este método es el llamado método "raster", donde cada elemento de área es asimilado a un punto de una imagen, que puede representar sólo un atributo. No se admiten aquí valores por "default" y todo punto tiene un valor. Otra variante es el método llamado "quadtree", donde la superfície de cada elemento de área es variable conforme la necesidad de representar con mayor o menor detalle, una porción del área total.

El segundo método importante, que llamaremos "vectorial", es más adecuado para representar objetos de características lineales. Los mismos son descritos usando listas de coordenadas correspondientes a los puntos de início y fin de los segmentos que componen el objeto. Un punto es representado por sus coordenadas, y un área a través de sus fronteras. Este método fue usado originalmente para representar datos urbanos, y su aplicación principal es la representación de mapas compuestos por polígonos disjuntos. Para este tipo de tarea hace un uso más eficiente de la memoria que el método anterior. Es también posible hacer, a traves de algoritmos, cálculos de la superfície, centro de gravedad y perímetro de un polígono, y determinar si un punto se encuentra dentro de un polígono.

EL MODELO RELACIONAL [2]

En el modelo relacional los datos son organizados usando relaciones. Se puede definir una relación de la siguiente manera: Sea una colección de conjuntos D_1 , D_2, ..., D_n(no necesariamente distintos), R es una relación de los conjuntos D_1, D_2, ..., D_n si es un subconjunto del producto cartesiano D_1 x D_2 x .... x D_n . El valor n es llamado el grado de la relación. Los conjuntos D_i son llamados domínios y los elementos de R, tuplas. El número de tuplas en una relación es la cardinalidad de la misma. Un atributo es el conjunto de valores extraídos de un domínio, que es usado en una relación. Se dice que una relación está normalizada o en primera forma normal cuando cada valor de un atributo es elemental, es decir, no admite descomposición. Un atributo de una relación es llamado clave primaria de esa relación, cuando los valores de ese atributo son diferentes para cada tupla de la misma. Representaremos una relación de la siguiente manera:

nombrerelación (Atributo1, Atributo2, ..., AtributoN)

Si una relación no tiene una clave primaria formada por un único atributo, siempre puede ser obtenida una clave primaria concatenando dos o más atributos. Si hay más de un atributo o combinación de atributos con la propiedad de poder ser usados como identificadores de tuplas, reciben en conjunto el nombre de claves candidatas. Aquellas claves candidatas que no son la clave primaria, son llamadas claves alternativas. Si un atributo de una relación es la clave primaria de otra, es llamado clave extranjera. Forman parte del modelo relacional las restricciones de integridad. Estas dicen que, en una relación, los valores de la clave primaria no pueden ser nulos (integridad de entidad) y que en el caso

de una relación tener una clave extranjera, cada valor de ese atributo tiene que aparecer como clave primaria de una tupla en una segunda relación (integridad referencial). Un banco de datos relacional es, entonces, un conjunto de relaciones normalizadas que cumplen las condiciones especificadas encima, con respecto a las claves y a la integridad.

IMAID [3]

Es un sistema que integra análisis de imágenes y gerencia de banco de datos. Fue desarrollado en la Universidad de Purdue con el propósito de almacenar imágenes de los satélites LANDSAT. Un mapa extraído de fotos satelitárias y compuesto por trazos rectilíneos (dibujo lineal) puede ser convertido en una relación con cuatro atributos especificando las coordenadas (x,y) de los puntos extremos de cada segmento. Pueden agregarse atributos adicionales si hubiera necesidad. Un punto es representado como un segmento de línea con sus dos extremos coincidentes. Una línea es representada por un conjunto de segmentos. Una región es representada por su frontera. Por ejemplo, en un mapa vial extraído de fotos satelitárias podríamos tener datos de rutas y ciudades. Cada foto, ruta y ciudad recibiría un número identificatorio: Nfoto, Nruta y Nciudad, respectivamente. Para cada segmento indicaríamos sus puntos extremos dentro de la foto y una ciudad sería tratada como una región. Ademas deseamos saber qué rutas y qué ciudades hay en cada foto, y la ubicación y otros datos de cada foto. Tendríamos entonces las siguientes relaciones:

rutas (Nfoto, Nruta, X1, Y1, X2, Y2)
nombreruta (Nfoto, Nruta, Nombre)
posición (Nfoto, Xtam, Ytam, Xcen, Ycen, Archivo)
ciudades (Nfoto, Nciudad, X1, Y1, X2, Y2)
nombreciudad (Nfoto, Nciudad, Nombre)

En la relación 'rutas', como puede haber más de un segmento para cada ruta, conseguimos la condición de unicidad de clave primaria, concatenando todos los atributos. Lo mismo ocurre con la relación 'ciudades'. Las relaciones 'nombreruta' y 'nombreciudad' tienen como clave primaria Nfoto-Nruta y Nfoto-Nciudad, dado que una ruta y una ciudad no pueden aparecer más de una vez en una foto. La relación 'posición' tiene como clave primaria Nfoto. Esto hace que el atributo Nfoto sea clave extranjera en las relaciones 'rutas', 'nombreruta', 'ciudades' y 'nombreciudad'. En las relaciones 'nombreruta' y 'nombreciudad' podemos tener la clave alternativa Nfoto-Nombre, si suponemos unicidad de los nombres.

GEO-QUEL [4]

Es una extensión al sistema de gerencia de banco de datos INGRES, desarrollado en la Universidad de Berkeley. Un mapa consta de una sola relación, donde cada tupla corresponde a un segmento de recta cuyos extremos están dados por los atributos X1, Y1, X2, Y2. Un punto es considerado como un segmento con extremos coincidentes. Regiones y líneas son considerados como agregaciones de segmentos y reciben un identificador (Idlínea e Idregión). Existe un atributo (Tipo) que indica si una tupla corresponde a un punto, un segmento aislado, una línea, una región o un punto que es el centro de una región. Además existen atributos destinados a especificar la manera de visualización de los datos. Esa unica relación respondería al esquema

mapa (X1, Y1, X2, Y2, Tipo, Vis1, Vis2, Idlínea, Idregión)

Vis1 y Vis2 son atributos de visualización. Otros atributos podrían agregarse.

Dado que un mapa es considerado como una colección de polígonos disjuntos, un segmento puede pertenecer a dos polígonos, o sea que las coordenadas no pueden ser consideradas como clave primaria. Una tupla siempre tiene un identificador de línea, pero puede tener un identificador nulo de región, asi que la combinación X1-Y1-X2-Y2-Idlínea sería la clave primaria de esta relación. No existen claves extranjeras al existir una sola relación.

DIMAP [5]

Este sistema combina un sistema de gerencia de banco de datos con un sistema de almacenamiento de imágenes y fue desarrollado en la Universidad de Illinois. Un banco de datos es un conjunto de mapas organizado jerárquicamente. Un mapa es un conjunto de relaciones, cada una correspondiendo a un tipo de objetos pictóricos. De cada una de estas relaciones se obtienen, por restricción, las relaciones correspondientes a un cuadro, que es la mínima unidad visualizable. Cada objeto pictórico corresponde a una tupla. Por ejemplo el mapa de una región podría estar formado por tres relaciones: 'ciudades', 'rutas' y 'ríos'. En un nivel inferior de la jerarquía, una ciudad incluída en esa región podría tener un mapa compuesto de dos relaciones: 'distritos' y 'población'. Para cada mapa existe una relación especial que explicita las relaciones componentes y el orden jerárquico de las mismas. Tambien existe para cada mapa una relación especial que indica el nombre de un

programa que grafica un cuadro de esa relación. En DIMAP los puntos son representados por sus coordenadas, una línea por sus extremos y un área es representada usando el metodo "raster", es decir, por sus puntos constituyentes.

Un mapa de zonas de vegetación sería representado por dos relaciones como:

vegetación (X, Y, Clase)
clasevegetación (Clase, Descripción)

En la relación 'vegetación' la clave primaria está formada por las coordenadas, mientras que Clase es una clave extranjera. Un mapa de puentes sería definido por una relación del tipo punto como:

puente (Nombre, X, Y, Tipopuente, ...)

Un mapa de rutas sería definido por una relación del tipo línea como

ruta (Nombre, X1, Y1, X2, Y2, Claseruta, ...)

En estas relaciones el nombre y/o las coordenadas forman la clave primaria. Claseruta y Tipopuente son claves extranjeras.

CONCLUSIONES

La primera conclusión que puede extraerse de la consideración de los sistemas expuestos es que, en el proyecto de un banco de datos geográficos, debe dedicarse gran atención a los aspectos de modelización, dado que la diversidad de enfoques encontrada indica que la complejidad de la aplicación es alta.

Si bien el método vectorial es más ampliamente usado, existen variantes en lo que respecta a la manera de agrupar en relaciones las coordenadas que representan los objetos y en cuanto al grado de explicitamiento con que éstos aparecen en el banco de datos. Por ejemplo, un punto puede constar de un par de coordenadas (x,y) o de dos pares idénticos de coordenadas, y una región puede aparecer como una entidad por derecho propio, es decir, en una relación de regiones, o sólo implícitamente, a través de un atributo con un valor común en todas las tuplas de segmentos que forman la frontera de la región.

El criterio de minimizar el número de relaciones influye en la fácilidad con que pueden ser agregados atributos a las entidades y hace que el esquema del banco de

datos sea bastante rígido. Un caso claro de esto es el sistema GEO-QUEL, donde un atributo da el significado de cada tupla en una relación única, sin permitir una caracterización diferenciada de cada objeto.

Otro problema de modelado está indicado por la aparición de claves primarias compuestas por varios atributos, en algunos casos abarcando toda la tupla, sin que esto se origine en relacionamientos entre entidades, como es lo normal.

La segunda conclusión que surje del análisis es que el modelo relacional, como fue expuesto, no permite expresar a través de sus mecanismos intrínsecos, dos características importantes de los datos geográficos. Estas son la agregación de objetos (v.g. segmentos para formar líneas) y el ordenamiento jerárquico dado por la inclusión de un objeto en otro (v.g. una ciudad en una región).

Las mismas son representadas mediante el uso de claves extranjeras, que en princípio deberían resolver problemas de dependencias funcionales, o sino recurriendo a relaciones especiales como la relación POT del sistema DIMAP, con atributos que tienen como valores nombres de relaciones, lo que no está permitido en el modelo relacional, ya que una relación es un valor compuesto.

En un proyecto en curso se están abordando estos temas con el objetivo de lograr una representación de los datos geográficos independiente de cualquier modelo de banco de datos y de determinar la utilidad de un modelo relacional expandido para la obtención de un esquema de bancos de datos geográficos.

REFERENCIAS

[1] Castro, H. D. "Concepts of geographic information systems". UFRGS-CPGCC, Porto Alegre, 1986.
[2] Date, C. J. "An introduction to database systems", Addison-Wesley, Reading, 1983.
[3] Chang, N. S. y Fu, K. S. "A relational database system for images" en "Pictorial information systems", Springer-Verlag, Berlin, 1980.
[4] Berman, R.R. y Stonebreaker, M. "GEO-QUEL: A system for the manipulation and display of geographic data". Computer Graphics 11(2), Summer 1977.
[5] Chang, S. K., Lin, B. S. y Walser, R. "A generalized zooming technique for pictorial database systems" en "Pictorial information systems", Springer-Verlag, Berlin, 1980.

UNA METODOLOGIA Y AMBIENTE DE PROGRAMACION DE SISTEMAS DISTRIBUIDOS A PARTIR DE REDES DE NUTT

M. Consuelo Franky de Toro
Universidad de los Andes
Bogotá - *Colombia*

1. INTRODUCCION

La programación de procesos de un sistema distribuido que va a operar sobre una red de computadores exige una metodología diferente a la que se utiliza para un sistema centralizado. En efecto, a diferencia de los procesos de un sistema centralizado, los procesos de un sistema distribuido se conciben como servidores permanentes y su sincronización se realiza por intercambio de mensajes y no por memoria compartida [Franky 85].

En la Universidad de los Andes se ha venido desarrollando y uniformizando una metodología de programación de sistemas distribuidos. En este artículo se presenta dicha metodología, la cual ya ha sido aplicada en la implementación de prototipos de investigación tales como los siguientes : sistema distribuido de transacciones bancarias, sistema de bibliotecas heterogéneas interconectadas, sistema manejador general de bases de datos distribuidas, etc.

La metodología de programación de sistemas distribuidos descrita en este artículo se basa en la utilización de la técnica gráfica de **"Redes de Nutt"**. Programar un sistema distribuido con esta metodología implica dos etapas :

- Especificar en redes de Nutt el conjunto de procesos que hacen parte del sistema, los cuales deben comunicarse únicamente a través de mensajes.

- Transformar las redes de Nutt de los procesos a un programa único que pueda ejecutarse en cada uno de los sitios (nodos) del sistema. Este programa debe simular el paralelismo de los procesos que van a ejecutarse en un mismo sitio.

Correspondiente a esta metodología se ha desarrollado en la Universidad de los Andes un software que facilita la realización de

las dos etapas mencionadas. Este software llamado "**Ambiente de Programación de Sistemas Distribuidos**" consta de dos módulos principales :

- Un Editor gráfico que permite describir y editar las redes de Nutt de los procesos del sistema distribuido que se quiere diseñar.

- Un Compilador que transforma las redes de Nutt previamente descritas con el Editor en un programa PASCAL que puede ejecutarse en cada uno de los microcomputadores de una red local particular. Este programa está estructurado de forma tal que permite la simulación de paralelismo de los procesos en computadores monoproceso (MS-DOS, CP/M).

A continuación se describe cada una de las etapas que implica la metodología de programación de sistemas distribuidos basada en Redes de Nutt.

2. ESPECIFICACION EN REDES DE NUTT DE LOS PROCESOS DE UN SISTEMA DISTRIBUIDO

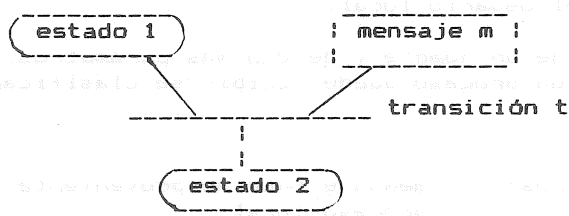
Las redes de Nutt [Nutt 73, Rolin 80, Franky 82] son una técnica gráfica derivada de las Redes de Petri [Peterson 77] que permite describir para cada estado de un proceso:

- los mensajes que puede recibir de otros procesos (locales o remotos),
- las acciones asociadas (acceso a datos locales, evaluación de condiciones o emisión de nuevos mensajes),
- el nuevo estado al cual pasa.

Esta técnica es muy adecuada para describir procesos distribuidos cuya sincronización no se realiza por memoria compartida sino por intercambio de mensajes. En este caso los procesos se consideran permanentes, a la espera de mensajes que los obligan a realizar acciones y pasar a esperar nuevos mensajes.

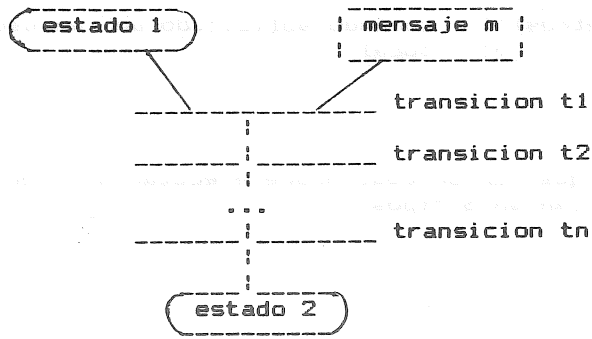
2.1 CONVENCIONES

Una Red de Nutt básica se representa con el siguiente grafo:

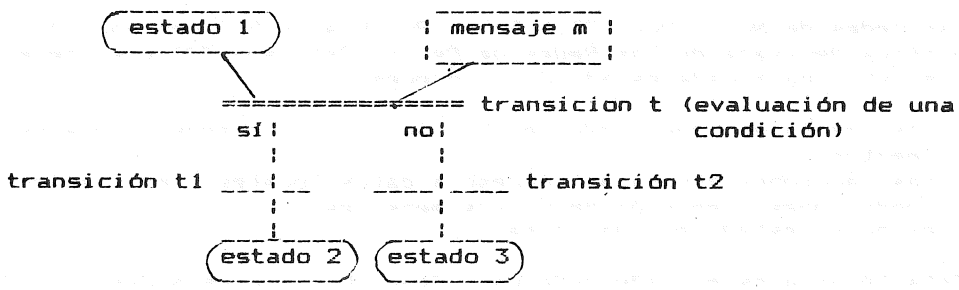


Cuando el proceso descrito se encuentra en el estado 1 y recibe el mensaje m, realiza las acciones asociadas a la transición t y pasa al estado 2.

En lugar de una sola transición t se puede considerar una serie de transiciones t1,...,tn que el proceso debe realizar al recibir el mensaje m :



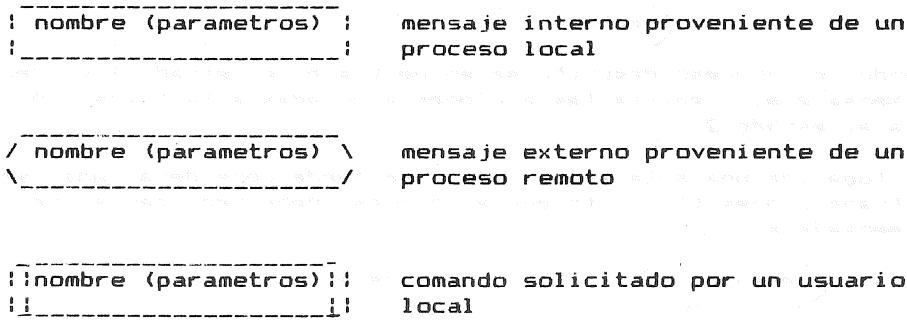
Además se pueden tener ramificaciones en una red de Nutt dependiendo de la evaluación de variables. Por ejemplo:



Un **proceso** se describe entonces a través de **un conjunto de redes de Nutt** que cubran todos los estados, mensajes que puede recibir y transiciones que debe realizar el proceso en cuestión.

Un proceso se ejecuta en un sitio específico dentro de un sistema distribuido y puede recibir mensajes de procesos locales (que se ejecutan en el mismo sitio) o de procesos remotos (que se ejecutan en sitios remotos). Si se trata de un proceso que interactúa con el usuario, puede recibir también mensajes (llamados comandos) provenientes del usuario local.

Cada mensaje consta de un nombre y de 0 o más parámetros.
Los mensajes que un proceso puede recibir se clasifican en 3 tipos:



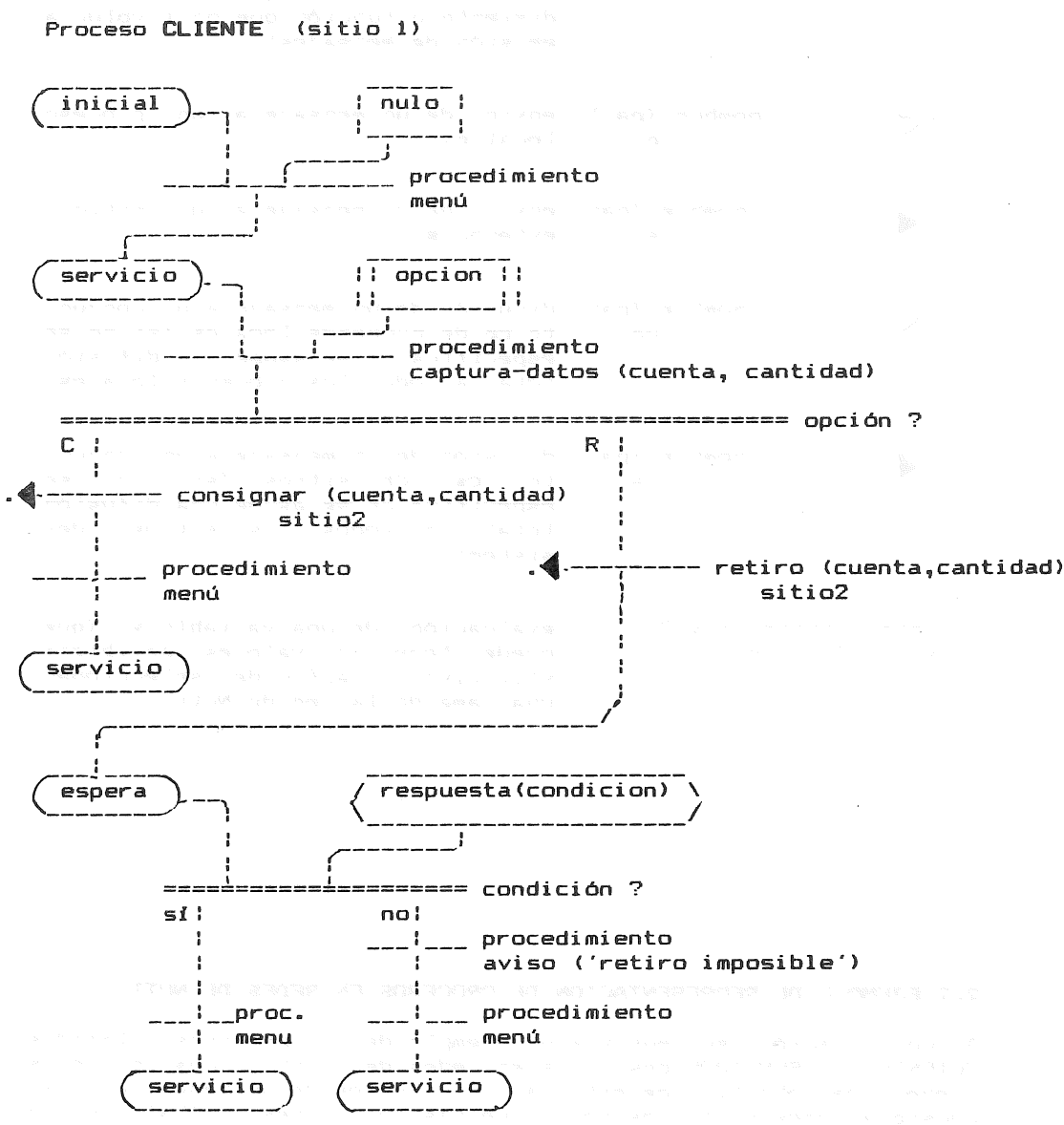
Las transiciones que un proceso puede ejecutar al recibir un mensaje se clasifican en 6 tipos:

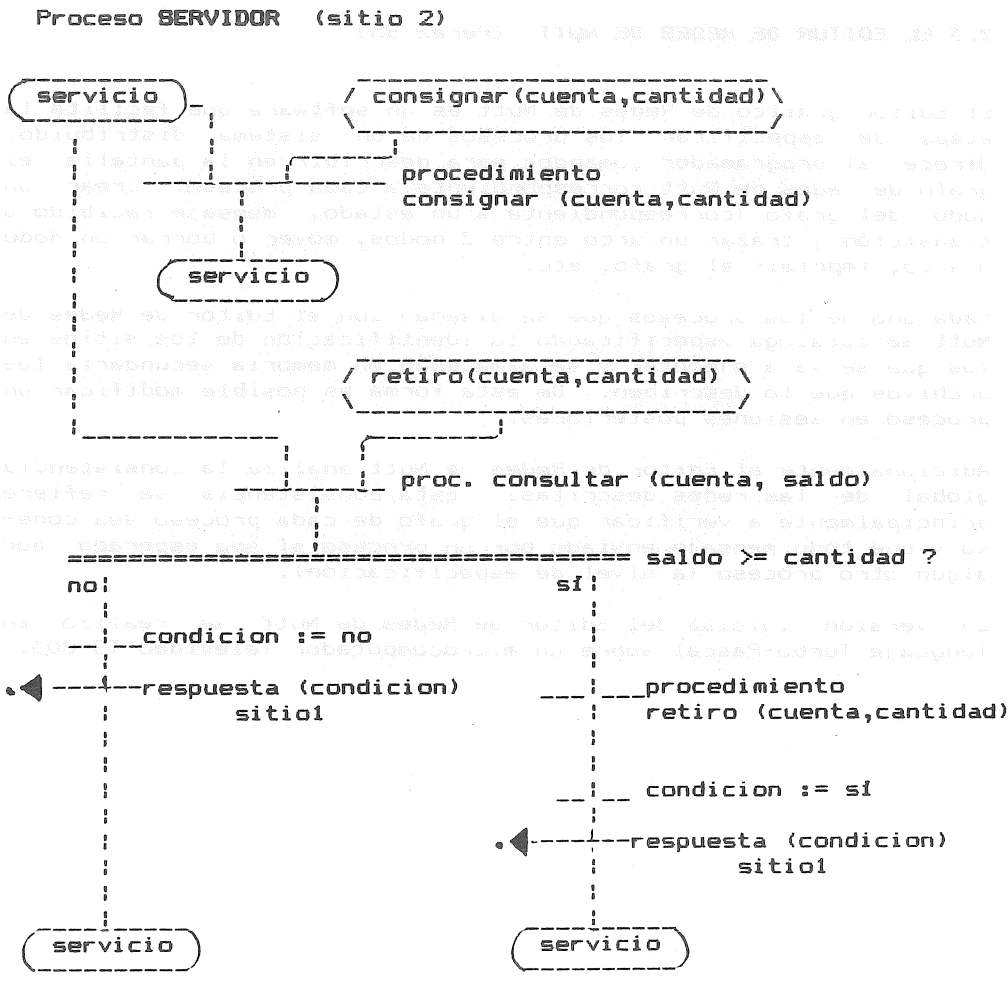
`--------` a	acción local a (asignación, procedimiento o función que no involucra emisión de mensajes)
`·◁--------` nombre (par) p	envio de un mensaje a un proceso local p
`·◀---------` nombre (par) s	envio de un mensaje a un sitio externo s
`···◁---------` nombre (par) cp	difusión de un mensaje a un conjunto cp de procesos locales (si no se especifica cp se asume una difusión total a todos los procesos locales)
`···◀---------` nombre (par) cs	difusión de un mensaje a un conjunto cs de sitios (si no se especifica cs se asume una difusión total a todos los sitios del sistema)

`========== v ?`
 `v1¦ v2¦...vn¦`
 `¦ ¦ ¦`
	evaluación de una variable v (que puede tener n valores posibles: v1,...,vn) a fin de seleccionar una rama de la red de Nutt

2.2 EJEMPLO DE REPRESENTACION DE PROCESOS EN REDES DE NUTT

A continuación se muestra un ejemplo de 2 procesos llamados CLIENTE y SERVIDOR descritos en redes de Nutt. Los procesos tienen como objetivo permitir a los clientes de un banco realizar consignaciones y retiros sobre cuentas corrientes: CLIENTE (en un sitio 1) recibe los pedidos e interactúa con los usuarios y SERVIDOR (en un sitio 2) actualiza las cuentas corrientes. Los 2 procesos se sincronizan intercambiándose mensajes :


Proceso CLIENTE (sitio 1)
inicial
nulo
procedimiento
menú
servicio
opcion
procedimiento
captura-datos (cuenta, cantidad)
opción ?
C
R
consignar (cuenta,cantidad)
sitio2
procedimiento
menú
retiro (cuenta,cantidad)
sitio2
servicio
espera
respuesta(condicion)
condición ?
sí
no
procedimiento
aviso ('retiro imposible')
proc.
menu
procedimiento
menú
servicio
servicio



Explicación: Inicialmente el proceso CLIENTE muestra un menú por pantalla para poder recibir las demandas del usuario (el mensaje especial "nulo" significa que el proceso no necesita recibir ningún mensaje para realizar esta transición inicial). Cuando el proceso CLIENTE recibe una demanda de consignación (opción C), la envía al sitio 2 (para que sea procesada por el proceso SERVIDOR) y sigue atendiendo nuevas demandas; cuando el proceso CLIENTE recibe una demanda de retiro (opción R) la envía al sitio 2, como en el caso anterior, pero antes de recibir nuevas demandas espera la respuesta correspondiente. Si el proceso SERVIDOR determina que un retiro no puede efectuarse (saldo insuficiente) envía al sitio 1 una respuesta negativa para que el proceso CLIENTE avise al usuario.

2.3 EL EDITOR DE REDES DE NUTT [Pérez 85]

El Editor gráfico de Redes de Nutt es un software que facilita la etapa de especificar los procesos de un sistema distribuido. Ofrece al programador comandos para describir en la pantalla el grafo de redes de Nutt correspondiente a cada proceso: crear un nodo del grafo (correspondiente a un estado, mensaje recibido o transición), trazar un arco entre 2 nodos, mover o borrar un nodo o arco, imprimir el grafo, etc.

Cada uno de los procesos que se diseñan con el Editor de Redes de Nutt se cataloga especificando la identificación de los sitios en los que se va a ejecutar y se almacenan en memoria secundaria los archivos que lo describen. De esta forma es posible modificar un proceso en sesiones posteriores.

Adicionalmente el Editor de Redes de Nutt analiza la consistencia global de las redes descritas. Esta consistencia se refiere principalmente a verificar que el grafo de cada proceso sea conexo y que todo mensaje enviado por un proceso sí sea esperado por algún otro proceso (a nivel de especificación).

La versión inicial del Editor de Redes de Nutt se realizó en lenguaje Turbo-Pascal sobre un microcomputador Televideo TS-803.

3. TRANSFORMACION DE LAS REDES NUTT DE UN CONJUNTO DE PROCESOS A UN PROGRAMA UNICO

La metodología de programación de sistemas distribuidos basada en Redes de Nutt implica, en una segunda etapa, transformar las redes de Nutt que describen los procesos del sistema diseñado a un programa único que pueda ser ejecutado en cada uno de los sitios del sistema.

El programa a generar debe simular la ejecución paralela de los procesos en cada uno de los sitios del sistema y debe incluir el código correspondiente a la interfaz con el nivel de comunicación de la red donde se va a implantar el sistema.

3.1 ESTRUCTURA DEL PROGRAMA A GENERAR

El programa se estructura en 3 partes o niveles:

- **nivel de Aplicación:** es el código correspondiente a los procesos descritos en redes de Nutt

- **nivel de Sistema Operacional Distribuido:** corresponde al núcleo principal que simula la ejecución paralela de los procesos del nivel de aplicación en un sitio del sistema

- **nivel de Comunicación:** es el código que permite la comunicación entre los procesos del nivel de aplicación ya sea dentro de un mismo sitio o entre sitios diferentes; este código depende de la red de comunicación específica sobre la cual se quiere implantar el sistema distribuido.

3.2 GENERACION DEL NIVEL DE APLICACION

Esta parte del programa contiene el código de ejecución que corresponde a cada proceso del sistema distribuido diseñado. Para ello se debe recorrer el grafo de redes de Nutt correspondiente a cada proceso y generar el siguiente esquema de código (suponiendo un lenguaje imperativo estilo PASCAL) :

```
Dependiendo-de estado haga:
¦
¦    e1 : Dependiendo-de mensaje-recibido haga:
¦         ¦    m11 : transiciones asociadas
¦         ¦          actualización de estado
¦         ¦    m12 : transiciones asociadas
¦         ¦          actualización de estado
¦         ¦    ...
¦         ¦    m1n : transiciones asociadas
¦         ¦          actualización de estado
¦         fin-dd
¦
¦    e2 : Dependiendo-de mensaje-recibido haga:
¦         ¦    m21 : transiciones asociadas
¦         ¦          actualización de estado
¦         ¦    m22 : transiciones asociadas
¦         ¦          actualización de estado
¦         ¦    ...
¦         ¦    m2n : transiciones asociadas
¦         ¦          actualización de estado
¦         fin-dd
¦   ....
¦
¦     em :Dependiendo-de mensaje-recibido haga:
¦         ¦   ....
¦         fin-dd
¦
fin-dd
```

Los estados del proceso están representados por **e1, e2,..., em.** Los mensajes que puede recibir el proceso en un estado **ei** están representados por **mi1, mi2,..., min.** En ejecución, cada vez que el proceso reciba un mensaje y obtenga turno examinará el estado en el cual está, ejecutará las transiciones asociadas y actualizará su estado.

La transformación general a código de cada uno de los tipos de transiciones de una red de Nutt es la siguiente:

- acción local: se debe transformar en una asignación o en una invocación del procedimiento asociado a la acción

- envío interno : se debe transformar en instrucciones para depositar el mensaje con la identificación del proceso destinatario en un área local llamada "Buzón" que es compartida por todos los procesos del sitio

- envío externo : se debe transformar en instrucciones para depositar el mensaje en una "Cola de Emisión" que maneja el nivel de comunicación; se indica el sitio destinatario

- difusión interna : se debe transformar en instrucciones para depositar en el Buzón un mensaje destinado a cada proceso local concernido

- difusión externa : se debe transformar en instrucciones para depositar el mensaje en la Cola de Emisión con la indicación de sus sitios destinatarios

- evaluación de una variable : se debe transformar en un "Dependiendo-de" (CASE en PASCAL).

Para ilustrar con un ejemplo, tomemos el caso del proceso CLIENTE presentado en la sección 2.2. Siguiendo el recorrido por profundidad del grafo del proceso se debe generar el siguiente código en PASCAL (el cual se muestra parcialmente):

```
.....

VAR EST-cliente : INTEGER ;        % variable que representa el
                                   % estado actual del proceso
.....
PROCEDURE cliente ;
BEGIN
  CASE EST-cliente OF
  ¦ 1 : BEGIN                      % estado inicial
  ¦     ¦ CASE MENSAJE-REC OF      % mensaje recibido
  ¦     ¦   0 : BEGIN              % 0 es el mensaje nulo
  ¦     ¦       ¦ menu ;
  ¦     ¦       ¦ EST-cliente := 2 ;
  ¦     ¦       END ;
  ¦     ¦ END ;
  ¦     END ;
  ¦
  ¦ 2 : BEGIN                      % estado servicio
  ¦     ¦ % en este estado se leen comandos del usuario
  ¦     ¦ .........                % lee un caracter en opcion
  ¦     ¦ IF KEYPRESSED THEN       % indica si se tecleo algo
  ¦     ¦    BEGIN
  ¦     ¦     ¦ captura-datos (cuenta, cantidad) ;
  ¦     ¦     ¦ CASE opcion OF
  ¦     ¦     ¦  'C':BEGIN
  ¦     ¦     ¦         ........ % meter mensaje de consig-
  ¦     ¦     ¦                  % nacion en cola de emision
  ¦     ¦     ¦           EST-cliente := 2 ;
  ¦     ¦     ¦       END ;
  ¦     ¦     ¦  'R': BEGIN
  ¦     ¦     ¦         ........ % meter mensaje de retiro en
  ¦     ¦     ¦                  % cola de emision
  ¦     ¦     ¦           EST-cliente := 3 ;
  ¦     ¦     ¦       END ;
  ¦     ¦     ¦ END ;
  ¦     ¦    END ;
  ¦     END ;
```

```
    ¦ 3 : BEGIN                    % estado espera
    ¦     ¦ CASE MENSAJE-REC OF
    ¦     ¦   1 : BEGIN
    ¦     ¦       ¦ condicion := COMPONENTE (1) ;
    ¦     ¦       ¦ % la funcion COMPONENTE extrae parametros
    ¦     ¦       ¦ % del mensaje recibido
    ¦     ¦       ¦ IF condicion = 'SI' THEN
    ¦     ¦       ¦    CONTROL := 1 ;
    ¦     ¦       ¦ IF condicion := 'NO' THEN
    ¦     ¦       ¦    CONTROL := 2 ;
    ¦     ¦       ¦ CASE CONTROL OF
    ¦     ¦       ¦   1 : BEGIN
    ¦     ¦       ¦         menu ;
    ¦     ¦       ¦         EST-cliente := 2 ;
    ¦     ¦       ¦       END ;
    ¦     ¦       ¦   2 : BEGIN
    ¦     ¦       ¦         aviso ('retiro imposible') ;
    ¦     ¦       ¦         menu ;
    ¦     ¦       ¦         EST-cliente := 2 ;
    ¦     ¦       ¦       END ;
    ¦     ¦       ¦ END ;
    ¦     ¦      END ;
    ¦     ¦ END ;
    ¦     END ;
  END ;
END ;
.........
```

Es de anotar que el tratamiento de los comandos provenientes del usuario en el proceso encargado de recibirlos (por ejemplo, el proceso CLIENTE) se debe hacer evitando la espera inactiva para no retardar indefinidamente el turno para los demás procesos. Para ello se debe dar turnos frecuentes a dicho proceso, y en cada turno el proceso esperará un tiempo prudencial máximo a que el usuario teclee un caracter para proceder o no a ejecutar las transiciones asociadas (por ejemplo, captura del comando completo con sus parámetros).

Adicionalmente, en este nivel de Aplicación se deben especificar los procedimientos y funciones nombrados en los grafos de redes de Nutt de los procesos.

Así por ejemplo para los procesos CLIENTE y SERVIDOR mostrados en la sección 2.2, el programador deberá especificar el texto de los procedimientos Consignar(cuenta, cantidad), Retiro(cuenta, cantidad), Menú, Captura-datos (cuenta, cantidad), Aviso(texto) y Consultar(cuenta, saldo).

3.3 GENERACION DEL NIVEL DE SISTEMA OPERACIONAL DISTRIBUIDO

Esta parte del programa debe simular la ejecución simultánea de los procesos descritos en redes de Nutt, encargándose entonces de de la activación y sincronización de dichos procesos.

Considerando solamente el caso en el que cada sitio del sistema distribuido es una máquina monoproceso, el nivel de Sistema Operacional Distribuido se constituye en el núcleo principal del programa que da turnos repetidos a los procesos del nivel de aplicación (invocando los procedimientos correspondientes) para simular su ejecución paralela.

En ejecución, el nivel de Sistema Operacional Distribuido trabaja examinando una **tabla de Estados-Mensajes** que especifica los estados de cada proceso y los mensajes que puede recibir un proceso en cada estado. Para el ejemplo del sistema distribuido de procesos CLIENTE y SERVIDOR presentados en la sección 2.2, la tabla de Estados-Mensajes tendría una información equivalente a la que se muestra a continuación :

PROCESOS	ESTADOS	MENSAJES
cliente	inicial	nulo
	servicio	opcion
	espera	respuesta(condicion)
servidor	servicio	consignar(cuenta,cantidad)
		retiro(cuenta,cantidad)

Teniendo en cuenta esta tabla, el estado actual de cada proceso **activo** y el Buzón donde los procesos depositan los mensajes que quieren intercambiarse, se decidirá en un momento dado a qué proceso se le puede dar turno de ejecución (será aquél que esté en el estado adecuado para recibir un mensaje del Buzón destinado a él). El Buzón se examina en el orden en que son depositados los mensajes. La unidad de turno de ejecución para un proceso le permitirá realizar las transiciones comprendidas entre la recepción de un mensaje para su estado actual y la actualización de ese estado.

Es importante anotar que el código que activa los procesos debe tener en cuenta los sitios en los que se va a ejecutar cada proceso; de esta forma, la ejecución del programa en un sitio dado debe activar solamente los procesos asignados a ese sitio.

Teniendo en cuenta las explicaciones anteriores, se puede presentar ahora el esquema general del programa a generar :

```
Programa
¦
¦ Declaración de variables usadas en las redes de Nutt
¦
¦ Declaración de variables internas usadas en el programa
¦
¦ Declaración de una variable estado para cada proceso
¦
¦ Declaración de procedimientos nombrados en las redes de Nutt
¦
¦ Declaración  de procedimientos del nivel de comunicación
¦
¦ Declaración  de un procedimiento por  proceso  conteniendo
¦ el código que se genera para cada proceso
¦
¦ Núcleo principal :
¦ ¦
¦ ¦  Cargar tabla de Estados-Mensajes en memoria principal
¦ ¦
¦ ¦  Inicializar variables internas
¦ ¦
¦ ¦  Inhibir procesos no asignados al sitio
¦ ¦
¦ ¦   Activar procesos que tienen un estado inicial asociado
¦ ¦   al mensaje "nulo" (la activación se hace mediante  in-
¦ ¦   vocación al procedimiento correspondiente al proceso)
¦ ¦
¦ ¦  Ciclo
¦ ¦    ¦
¦ ¦    ¦ Dar turno al proceso encargado de recibir los coman-
¦ ¦    ¦ dos del usuario
¦ ¦    ¦
¦ ¦    ¦ Dar  turno  al  proceso destinatario de uno  de  los
¦ ¦    ¦ mensajes del buzón teniendo en cuenta la tabla Esta-
¦ ¦    ¦ dos-Mensajes y el estado actual de cada proceso
¦ ¦    ¦
¦ ¦   fin-ciclo
¦ fin-nucleo-principal
fin-programa
```

3.4 GENERACION DEL NIVEL DE COMUNICACION

El programa debe incluir en su parte de nivel de comunicación los procedimientos que permiten el intercambio de mensajes entre sitios en la red sobre la cual va a operar el sistema distribuido.

Estos procedimientos deben encargarse de extraer cada mensaje de la Cola de Emisión de un sitio para conformar paquetes que puedan transmitirse por la red a otro sitio. Así mismo deben recibir los paquetes que llegan al sitio, y deducir de ellos los mensajes que deben depositarse en el Buzón local.

Los procedimientos del nivel de comunicación deben corresponder a

los protocolos de comunicación vigentes para la red específica con la cual se está trabajando.

3.5 COMPILADOR DE REDES DE NUTT [Del Risco 85]

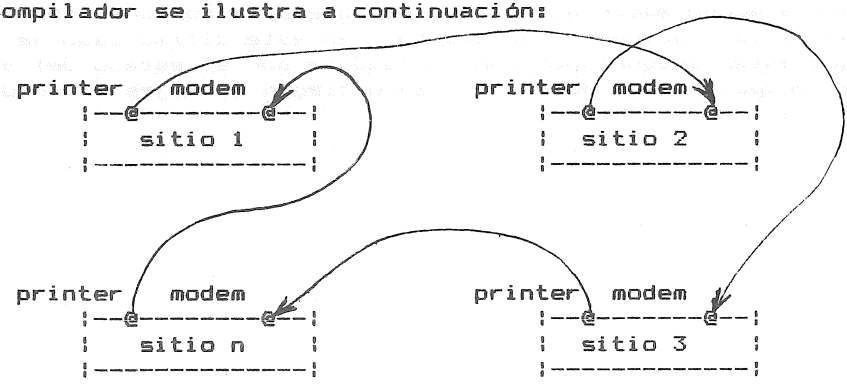
El Compilador de Redes de Nutt es un software que automatiza la etapa de transformar a un programa las redes de Nutt de un conjunto de procesos, especificadas previamente con el Editor de Redes de Nutt.

En su funcionamiento el Compilador de Redes de Nutt recorre en forma recursiva las redes de Nutt de los procesos generando el código en Turbo-Pascal correspondiente a cada uno de ellos. El programador debe completar la especificación en Turbo-Pascal del tipo de variables utilizadas y de los procedimientos y funciones incluidas en las redes de Nutt. El Compilador produce entonces un programa global que da turnos a cada uno de los procesos para simular su ejecución paralela.

Las partes del programa llamadas nivel de Aplicación y nivel de Sistema Operacional Distribuido corresponden a los esquemas generales expuestos en las secciones precedentes.

Como nivel de Comunicación, el Compilador incluye en el programa los procedimientos que permiten la comunicación de microcomputadores en una red local simple con topología de anillo unidireccional. La versión inicial de este Compilador fué desarrollada para microcomputadores Televideo TS-803 (sistema CP/M).

La topología de la red local asumida por el nivel de Comunicación del Compilador se ilustra a continuación:



Cada micro se conecta a otros dos (sucesor y predecesor) a través de sus puertos seriales MODEM y PRINTER; por el PRINTER un micro envía sus mensajes a su sucesor y por el MODEM los recibe de su predecesor.

El nivel de Comunicación funciona en base a un mecanismo de interrupción aprovechando las facilidades que para ello provee el STI

[STI] encargado de controlar el puerto MODEM. El funcionamiento de este nivel se puede describir así:

Constantemente circula por la red un mensaje de control llamado "Ficha" (es generado la primera vez por el sitio 1 cuando se pone en funcionamiento la red). Un sitio solo puede transmitir los mensajes de su Cola de Emisión cuando tiene la Ficha en su poder (los mensajes colocados en esta cola corresponden a los envíos externos que desean hacer los procesos locales). La llegada de la Ficha a un sitio genera una interrupción con las siguientes acciones:

- recepción de todos los mensajes que vengan después de la Ficha provenientes del sitio predecesor en el anillo (un primer mensaje de control indica cúantos mensajes se deben recibir); estos mensajes son colocados en la Cola de Emisión del sitio

- verificación del destino de cada mensaje de la Cola de Emisión: si el destino coincide con el sitio, el mensaje es colocado en el Buzón local (este es el buzón que examina el nivel de Sistema Operacional Distribuido para dar turnos de ejecución a los procesos locales); si el destino no es el sitio, el mensaje se deja en la Cola de Emisión

- emisión al sitio sucesor de la ficha y de los mensajes de la Cola de Emisión (dicha cola queda finalmente vacía).

Es importante resaltar que la implantación del Nivel de Comunicación en base a interrupciones lo hace totalmente transparente a los demás niveles del programa generado y además, asegura una eficiencia mucho mayor que la que se obtendría con una implantación sin interrupciones: en efecto, en este último caso se tendría que estar preguntando constantemente por el estado del puerto de recepción para detectar cuando llegan mensajes y entonces recibirlos.

4. CONCLUSIONES

La motivación principal para usar la técnica gráfica de las Redes de Nutt es que permite programar procesos distribuidos de forma mucho más natural que lo que permite la algorítmica convencional (la sincronización de estos procesos se hace por mensajes y no por memoria compartida). Su simplicidad, la visión general que presentan de todo el sistema diseñado, de cada uno de los procesos que lo constituyen y de la comunicación entre ellos, son algunas de las ventajas que ofrecen las Redes de Nutt.

La metodología de programación de sistemas distribuidos expuesta en este artículo indica cómo representar procesos en redes de Nutt y cómo transformar esas redes a un programa que pueda ejecutarse en cada uno de los sitios del sistema. La metodología indica cómo generar código correspondiente a las partes del programa llamadas niveles de Aplicación y de Sistema Operacional Distribuido dejando abierta únicamente la parte del nivel de Ccomunicación, la cual depende estrictamente de la red sobre la cual se está trabajando.

Adicionalmente, el ambiente de programación constituido por el Editor y el Compilador de Redes de Nutt constituye una herramienta poderosa para la programación de sistemas distribuidos. Con ella se pueden programar en principio protocolos de comunicación para niveles superiores, protocolos de ejecución de transacciones para aplicaciones distribuidas, o sistemas distribuidos completos contemplando varios niveles (por ejemplo, un sistema de Base de Datos Distribuida).

Con este ambiente de programación distribuida el programador se concentra en el diseño de cada uno de los procesos de su sistema, despreocupándose de los aspectos de comunicación entre sitios y de la simulación de la ejecución paralela de los procesos en cada sitio : estos aspectos son resueltos de forma automática por los niveles de Comunicación y de Sistema Operacional Distribuido incluidos en el programa generado por el Compilador de Redes de Nutt.

Actualmente el ambiente de programación distribuida tiene limitaciones por memoria ya que fue implantado en un ambiente de microcomputadores CP/M de 64K. Se proyecta en un futuro transportarlo a un ambiente de equipos más potentes para eliminar dichas limitaciones.

Eventualmente el ambiente de Programación Distribuida descrito en este artículo puede aprovecharse para simular el funcionamiento de sistemas distribuidos que vayan a ser implantados no solo en redes locales sino también en redes de gran alcance, reduciendo así los costos de las pruebas iniciales del sistema.

BIBLIOGRAFIA

[Del Risco 85] : José M. Del Risco, Antonio J. Del Risco
"Simulador e Interpretador de Redes de Nutt para aplicaciones distribuidas", Proyecto Dirigido, Dpto. de Ing. de Sistemas, Universidad de los Andes, Colombia, Julio de 1985.

[Franky 82] : M. Consuelo Franky de Toro
"Contribution au controôle de cohérence des systèmes transactionnels répartis en vue de favoriser l'utilisateur intéractif", Tesis de Doctorado de Tercer Ciclo en Informática, Univeridad de Lille I, Francia, Enero de 1982.

[Franky 85] : M. Consuelo Franky de Toro
"Notas del Curso de Sistemas Distribuidos", Dpto. de Ing. de Sistemas, Universidad de los Andes, Colombia, 1985.

[Nutt 73] : J. D. Noe, G. J. Nutt
"Macro E-Nets for representation of parallel systems", IEEE Transactions on Computers, Volume C-22, No. 8.

[Pérez 85] : Daniel Pérez M.
"Editor y Verificador de consistencia para Redes de Nutt", Tesis de Postgrado, Dpto. de Ing. de Sistemas, Universidad de los Andes, Colombia, Agosto de 1985.

[Peterson 77] : J. L. Peterson
"Petri Nets", ACM Computing Surveys, Vol. 9, No.3, Sept. 1977.

[Rolin 80] : P. Rolin
"Exploitation d'un modèle d'evaluation de Nutt au cours de la vie d'un produit", INRIA Projet SIRIUS, 1980.

[STI] :
"Z80 Microcomputer Peripherals - S.T.I. Controller", MK3801.

MODELADO AUTOMATICO DE DATOS Y "QUERY LANGUAGES" INTELIGENTES

Breogán Gonda Vázquez
Juan Nicolás Jodal
São Paulo - *Brasil* - Montevideo - *Uruguay*

I. Introducción

En los últimos años el uso de bancos de datos soportados por sistemas de gerencia de banco de datos ha aumentado mucho. Los resultados, sin embargo, muchas veces no corresponden a las expectativas.

Al principio los sistemas de gerencia de banco de datos fueron utilizados cuando se trataba de administrar acervos de datos de gran volumen o de estructura compleja o que necesitaban múltiples vías de acceso. Actualmente su uso se ha generalizado buscando fundamentalmente aumentar la productividad del desarrollo de sistemas y disminuir sus costos de mantenimiento.

Sin duda la baja productividad en el desarrollo de sistemas y los altos costos de mantenimiento son problemas importantes de la informática y vistos relativamente su ponderación sube cada día con relación a los de eficiencia del uso de los recursos de hardware por el gran abaratamiento relativo de estos. Ante esta situación parece especialmente inadecuado tratar de representar los datos pensando en una "eficiencia" que no conseguimos definir bien mientras que, muchas veces, no logramos la imprescindible eficacia.

Estas consideraciones nos orientan a una representación más objetiva del dato.

Debemos, sin embargo, considerar otros elementos: son realmente el costo de mantenimiento y la baja productividad de desarrollo de sistemas los problemas principales de la informática en las empresas?. Estos son, sin duda, problemas importantes pero no son los esenciales: las empresas, de algún modo, tienen cubiertas sus necesidades operativas pero la información de apoyo a las decisiones tácticas y estratégicas es muy inadecuada.

O sea que no estamos dando el debido soporte a las

decisiones más riesgosas.

Cabe preguntarse si es posible implementar sistemas de informacíon que den un buen soporte a estas decisiones.

Sin duda es posible implementar sistemas de información para soportar las actividades operativas, pero estas son previsibles y, en consecuencia, estructurables. Lo son las actividades de toma de decisiones tácticas y estratégicas? ciertamente debemos concluir que en muchos casos no lo son. Entonces, como podremos resolverlas por medio de sistemas?.

El elemento esencial para proveer a la empresa la información necesaria es la "usabilidad" del dato. Necesitamos bancos de datos tales que si existen en ellos los datos necesarios para derivar una determinada información esa derivación sea posible a bajo costo. Este "bajo costo" necesitamos medirlo fundamentalmente en tiempo: tiempo de proceso pero sobre todo tiempo de formulación del pedido de información ya que casi siempre estamos respondiendo a una necesidad no previsible.

Esto nos lleva a la necesidad de modelos que representen fielmente la realidad y nos aseguren una gran "usabilidad" de los datos. Sería ideal pensar en una implementación directa de estos modelos pero las condiciones actuales del ambiente software/hardware a veces nos obligan, por ineficiencias de este ambiente, a introducir desvíos. Eso es totalmente cierto: nuestra experiencia profesional nos dice que en todos los bancos de datos en que hemos trabajado debimos introducir algún desvío para hacer frente a algunos factores críticos pero también nos dice que su importancia relativa, medida en cantidad de tablas con desvíos sobre cantidad total de tablas, es muy pequeña.

Esto nos sugiere manejarnos con un modelo teoricamente ideal e introducir los desvíos que sean estrictamente necesarios.

Vista, entonces, la especial importancia del modelo, falta encontrar el camino para construirlo.

Existen varias metodologías para la obtención de modelos con diferentes orientaciones. Es posible obtener un buen modelo basándonos sólo en elementos formales?. Es implementable en las condiciones software/hardware de hoy o previsibles en el corto plazo un modelo basado en elementos semánticos?.

Un modelo que sólo considera elementos formales puede ser insuficiente para una representación fiel mientras que un modelo que coloca el énfasis en elementos semánticos tiende a complicarse y a hacerse subjetivo. Donde está el equilibrio?.

El modelado de datos no es una tarea individual sino una

tarea de equipo donde intervienen usuarios, analistas de aplicaciones, administradores de datos y administradores de banco de datos. En consecuencia debemos tener un procedimiento claro y simple porque de otra manera el resultado de este trabajo de equipo es imprevisible.

En este trabajo exponemos un procedimiento que cumple con estas condiciones.

La existencia del procedimiento viabiliza la concreción de un programa que efectúe las tareas de modelado en forma incremental.

La existencia de un modelo totalmente representado en forma computacional permite "query languages inteligentes" que sustituirán y/o complementarán a los actuales para el acceso a los bancos de datos relacionales y a los, cada vez más importantes, bancos de datos públicos.

II. Reglas propuestas

Un primer elemento que sirve de antecedente a este trabajo está constituído por las investigaciones del Dr. Edgar F. Codd de I.B.M., realizados en fines de la década del 60 y principio de la del 70.

Codd introduce lo que llamaremos Modelo Relacional Básico que consiste fundamentalmente en lo siguiente:

a) Representación simple: Representación según un conjunto de tablas

b) Procedimiento para obtener representaciones de los datos que no contengan redundancia lógica: normalización

c) Operadores poderosos para tratar las tablas: álgebra relacional

d) Restricciones de integridad para la consistencia intra-tablas

Podemos considerar aquí los siguientes elementos:

d1) Las filas de una tabla son todas diferentes y, en consecuencia :

d2) Existe por lo menos una super-llave que es cualquier identificador unívoco de una fila entre todas las de una tabla

d3) Existe por lo menos una llave candidata que es cualquier super-llave mínima (una super-llave tal que ningún sub-conjunto estricto de ella sea una super-llave)

d4) Llave que es la llave candidata elegida en el modelo para una determinada tabla

d5) Ningún atributo de la llave puede tomar valores nulos

d6) El valor que toma el atributo - simple o compuesto - llave, en una determinada fila de una tabla no puede modificarse

Todo esto constituye una buena base y además está implementado en la mayor parte de los sistemas de gerencia de

banco de datos autoproclamados como "relacionales" que están hoy en el mercado, pero es insuficiente ya que no nos asegura un modelo consistente.

Cuales son los problemas?. El esencial es que en este modelo las tablas son independientes, lo que no es cierto en la realidad.

El problema más visible se soluciona con las siguientes reglas:

e) Si x, atributo simple o compuesto, es llave de una tabla T1 lo llamaremos "primario" y puede aparecer también en otras tablas. Si y, atributro simple o compuesto, es tal que no existe en el modelo una tabla que lo tenga como llave, lo llamaremos "secundario" y no puede aparecer en ninguna otra tabla.

f) No existirán dos tablas con la misma llave

g) Si x, atributo simple o compuesto, es llave de una tabla T1 y aparece también en una segunda tabla T2, dada una fila cualquiera de T2, j, el valor correspondiente de x, T2.xj debe ser tal que exista en T1 una fila i tal que el valor correspondiente de x, T1.xi cumpla T1.xi = T2.xj.
(∀ x EX T2.x ===> EX T1.x)

En este caso decimos, más brevemente que T2 está subordinada a T1 según T2.X ===> T1.X.

Esto, sin embargo, es de utilidad en un ambiente padronizado de nombres y el modelo relacional básico no lo es.

Adicionalmente, hasta el momento, el significado del modelo reside en el conjunto de los nombres de las tablas y los nombres de los atributos, lo que constituye un casuismo bastante indeseable.

En particular es causante del bajo nivel de los "query languages" basados en álgebra relacional, como se muestra a continuación:

```
select CLIENTE.CLIENTE#, NOMBRE_CL, FACTURA.FACT#,
       FECHA-FACT,      LINEA_FAC.PRODUCTO#,     DESC_PR,
       CANT_VEND, PU_VEND, CANT_VEND * PU_VEND
from   CLIENTE, FACTURA, LINEA_FAC, PRODUCTO
where  CLIENTE.CLIENTE# = FACTURA.CLIENTE#
and    FACTURA.FACT# = LINEA_FAC.FACT#
and    LINEA_FAC.PRODUCTO# = PRODUCTO.PRODUCTO#
 and    condiciones del problema
           ..........................
           ..........................
           ..........................
```

La causa del bajo nivel de estos lenguajes reside en que el sistema "no conoce" el modelo, dada su casuística, y es el usuario que debe sustituirlo en esta tarea.

Una primera regla de padronización es la siguiente:

h) Si dos atributos tienen el mismo "significado" deben tener el mismo nombre y si tienen diferente "significado" deben tener diferente nombre, **siempre que ello sea posible.**

i) No existirán en una tabla dos atributos con el mismo nombre.

Esta regla introduce un primer elemento semántico que es el "significado" que podríamos caracterizar como la "naturaleza del dato". Notese que lo caracterizamos y no lo definimos: apelamos a nuestros conceptos primitivos.

Esta dificultad para definir debe ser contornada a la hora de especificar porque, de lo contrario, introduciremos una fuerte subjetividad en el modelo. Para ello, introduciremos el concepto de dominio asociado a un atributo y el de "familia de atributos". El "dominio" consistirá, para nosotros, en un conjunto de valores posibles, dado explicita o implicitamente, un formato de esos valores y el significado del atributo. Un atributo tiene asociado un determinado dominio del que toma sus valores.

Diremos que los atributos **a** y **b** forman una familia si se cumple:

1. **a** y **b** son equivalentes o bien

2. **a** es una generalización de **b** o (lo que es equivalente) **b** es una particularización de **a**

Diremos que **a** es una generalización, en sentido amplio, de **b** si se cumple (1) o (2).

Podrá darse el caso de que más de un atributo tome valores del mismo dominio?. Si, supongamos el caso de que existe una tabla cuya llave es **producto#** y otra cuya llave es **producto_compuesto# + producto_componente#** aquí el significado de **producto#**, **producto_compuesto#** y **producto_componente#** es el mismo pero se da lo siguiente:

- por (i) los atributos **producto_compuesto#** y **producto_componente#** no pueden tener el mismo nombre

- todo **producto-compuesto#** tiene asociado un **producto#**, pero el inverso no es cierto (**producto** es una generalización en sentido estricto de **producto-compuesto**)

- simetricamente, todo **producto-componente#** tiene asociado un **producto#**, pero el inverso no es ciento (**producto** es una generalización en sentido estricto de **producto-componente**)

Como regla práctica usaremos la siguiente:

- si existe un único atributo con un determinado significado, daremos a atributo, dominio y significado el nombre del atributo

- si existen varios atributos con el mismo significado, debemos determinar si existe entre ellos una relación de generalización/particularización la cual pasará a integrar el modelo.

ATRIBUTO	PARTICULARIZACI'ON DEL ATRIBUTO
b	b
a	b

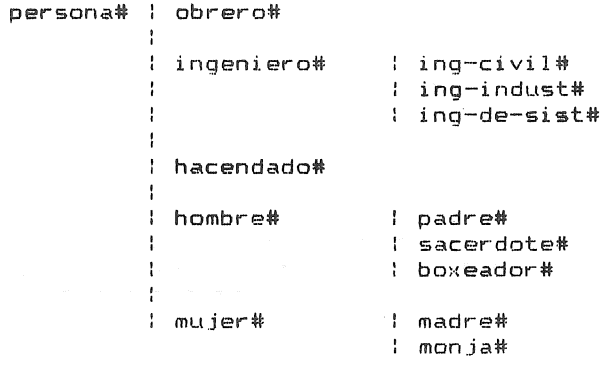
lo que deja claramente expresada la relación existente.

Véase que a esta altura surge claro el "siempre que sea posible" de (h), pero, con esa frase, la utilidad de (h) es bastante relativa.

Adicionalmente, lo que hemos visto para dos niveles, es válido para cualquier cantidad de niveles:

ATRIBUTO	PARTICULARIZACION DEL ATRIBUTO
persona#	persona#
obrero#	persona#
ingeniero#	persona#
hacendado#	persona#
ing-civil#	ingeniero#
ing-indust#	ingeniero#
ing-de-sit#	ingeniero#
hombre#	persona#
mujer#	persona#
padre#	hombre#
sacerdote#	hombre#
boxeador#	hombre#
madre#	mujer#
monja#	mujer#

lo que corresponde a la jerarquía:



Debemos re-escribir (h), de la siguiente forma:

j) Si dos atributos tienen significados diferentes tendrán nombres diferentres.

k) Si varios atributos tienen el mismo significado se dará el mismo nombre a aquellos que sean equivalentes y, en otros casos, se darán nombres que representen claramente las relaciones de generalización o particularización, directas o transitivas, que existan entre ellos.

Ahora, podemos re-formular (g) de la siguiente forma:

l) Si el atributo, simple o compuesto, x = x0,x1,x2,...,xn es llave de una tabla T1 y existe en una tabla T2 el atributo, simple o compuesto, y = y0,y1,y2,...,yn tal que, para todo i se cumple que xi es una generalización en sentido amplio de yi, por razones de consistencia, debe cumplirse que, ∀ i EX T2.yi ==> EX T1.xi siendo xi = yi, o más brevemente, diremos que T2 esta subordinada a T1 según T2.y ===> T1.x.

De esta forma estamos resolviendo los principales problemas de consistencia.

En estas condiciones el significado del modelo reside en los nombres de los atributos, siendo los nombres de las tablas irrelevantes.

Además, ahora el sistema "conoce" el modelo y podemos pensar en problemas como el del "query" que ya hemos visto y cuya solución basada en álgebra relacional listamos a continuación, y resolverlo ahora de una forma mucho más elegante y simple mediante cálculo relacional:

- Solución tradicional, mediante álgebra relacional

```
select CLIENTE.CLIENTE#, NOMBRE_CL, FACTURA.FACT#,
       FECHA-FACT,     LINEA_FAC.PRODUCTO#,     DESC_PR,
       CANT_VEND, PU_VEND, CANT_VEND * PU_VEND
from   CLIENTE, FACTURA, LINEA_FAC, PRODUCTO
where  CLIENTE.CLIENTE# = FACTURA.CLIENTE#
and    FACTURA.FACT# = LINEA_FAC.FACT#
and    LINEA_FAC.PRODUCTO# = PRODUCTO.PRODUCTO#
and    condiciones del problema
          ..........................
          ..........................
          ..........................
```

Solución mediante cálculo relacional

```
select CLIENTE#,    NOMBRE_CL,    FACT#,    FECHA-FACT,
       PRODUCTO#, DESC_PR, CANT_VEND, PU_VEND,
       CANT_VEND * PU_VEND
where  condiciones del problema
          ..........................
          ..........................
          ..........................
```

Aún queda, sin embargo, un problema para resolver: como conoce el usuario los atributos?

Como ahora el sistema "conoce" el modelo, podemos pensar en un query guiado de mucho más alto nivel, desde el punto de vista del usuario.

Adicionalmente, podemos ver que la operación de JOIN es inconsistente y de resultados imprevisibles a menos que se cumpla, por lo menos, lo siguiente:

m) La fusión (JOIN) entre dos tablas T1 y T2 según los atributos, simples o compuestos, x = x0,x1,x2,...,xn de la tabla T1 y y = y0,y1,y2,...,yn de la tabla T2 sólo puede tener sentido si se cumple que, para todo i es significado(xi) = significado(yi).

En rigor, en las mayor parte de los casos, este operador no

tendrá utilidad alguna si no se cumple lo siguiente, que es bastante más restrictivo:

n) La fusión (JOIN) entre dos tablas t1 y t2 según los atributos, simples o compuestos, x = x0,x1,x2,...,xn de la tabla T1 y y = y0,y1,y2,...,yn de la tabla T2 tiene sentido si se cumple que, o bien T2 está subordinada a T1 según T2.y ===> T1.x, o bien que T1 está subordinada a T2 según T1.x ===> T2.y.

Adicionalmente, para cumplir con (l) de una forma viable en las condiciones software/hardware de hoy, todos los atributos primarios y aquellos secundarios que sean particularizaciones de atributos primarios deberán ser indexados en cada una de las tablas en que aparezcan (a menos que la subordinación entre las tablas en que se hallen involucrados esté asegurada por transitividad) lo que hace que, en casi todos los casos, las navegaciones consistentes en el banco de datos puedan hacerse aprovechando índices que ya existen por razones de consistencia.

III. Resultados

Las reglas enunciadas nos dan un procedimiento simple y objetivo para el modelado de datos.

Cuando el tamaño de los modelos crece, la dificultad crece mucho más, con independencia de la metodología utilizada.

La metodología propuesta, dado que esta basada en reglas claras, tiene la ventaja adicional de que puede utilizar con mucho provecho herramientas automáticas.

Estas herramientas automáticas pueden ayudarnos en lo siguiente:

- Creación y mantenimiento del modelo totalmente normalizado por síntesis, a partir de visiones objetivas de los datos.

- Introducción y mantenimiento de desvíos al modelo totalmente normalizado, las que, junto con aquel, representan el modelo físico del banco de datos.

- Pre-procesador de lenguaje "query" basado en cálculo relacional generando codigo fuente para procesadores basados en álgebra relacional.

- Pre-procesador de "query" guiado de alto nivel generando codigo fuente para procesadores basados en álgebra relacional.

- Generador de visiones y lógica de acceso al, y de mantenimiento de consistencia del, banco de datos para lenguajes de tercera y cuarta generación.

- Generador de aplicaciones extremadamente poderoso.

- Esto viabiliza una nueva generación de sistemas de gerencia de banco de datos, los que llamaremos "relacionales inteligentes" que "conocen" su modelo que incluye los siguientes puntos:

  a) Representación tabular

  b) Normalización

  c) Operadores del álgebra relacional

  d) Integridad intra-tablas

  e) Cada atributo secundario aparece en una única tabla del modelo

  f) No existencia de dos tablas con la misma llave

  g) Subordinación (esta condición es un caso particular de (l))

  h) Padronización de nombres de atributos (esta condición es un caso particular de (k))

  i) No existencia en una determinada tabla de dos atributos con el mismo nombre

  j) Atributos con significados diferentes tendrán nombres diferentes

  k) Padronización generalizada y jerarquización de atributos con el mismo significado

  l) Subordinación generalizada

  m) Fusión consistente

Bibliografía

Codd, E. F. [1970] A Relational Model of Data for Large Shared Data Banks, CACM, Junio 1970.

Codd, E. F. [1979] Extending the Database Relational Model to Capture More Meaning ACM TODS, Diciembre 1979.

Date, C.J. [1981] An Introduction to Database Systems, 3ra. Ed.,Addison-Wesley

Maier, D. [1983] The Theory of Relational Databvases, Pitman

SCI [1986] Data Expert: Manual del Usuario

SCI [1986] Data Expert: Manual de Referencia

Gonda, B. y Jodal, N. [1986] Modelado de Datos, I Seminario Latino-Americano de Administración de Datos, Petrópolis, Rio de Janeiro, Brasil, Junio de 1986

SISTEMA EXPERTO PARA EL DIAGNOSTICO DE FALLAS

Autores: *Alejandro Loccicero, Guillermo Vázquez, José Aronson*
Coordinación y dirección: *Jorge Pluss, Manuel Molina*
Universidad Nacional de Rosario
Rosario - *Argentina*

RESUMEN

El SEDIF (Sistema Experto para el Diagnóstico de Fallas) es un Programa Inteligente que permite, entre otras cosas, detectar fallas en Sistemas Electrónicos mediante el suministro de datos relevantes sobre los síntomas que presentan cuando se observa un mal funcionamiento en los mismos.

La característica principal de este Sistema radica en el hecho de que su comportamiento se asemeja al de un experto humano en diagnóstico de fallas, logrado con el empleo de técnicas de la Inteligencia Artificial (IA), una disciplina formal de la Informática.

Así mismo se describe el diseño de un prototipo sobre el cual se realiza la prueba del Sistema Experto (SE), este es un Controlador de Acceso a un Recinto (CAR).

INTRODUCCION

El problema a resolver consiste en diseñar un Sistema Computable que manifieste las siguientes características:

1- Toma de Decisiones
Deberá llevar el control permanente durante la búsqueda de soluciones a través de su Sistema Ejecutivo, interrogando al usuario cuando no pueda continuar con los datos suministrados inicialmente, minimizando en lo posible el número de consultas a realizar al mismo, apelando a técnicas de control apropiadas. Esto es así puesto que entendemos que el máximo esfuerzo lo debe realizar SEDIF, lo que tiene más sentido si pensamos que el operador no conoce absolutamente nada sobre Electrónica.

2- Interacción Usuario-Sistema Clara y Simple
La forma en que presenta los resultados debe ser explícita, de manera que cualquier persona pueda interpretar a SEDIF. Asimismo, las respuestas que el usuario debe darle tendrán un formato sencillo.

3- Capacidad Explicativa
Será necesario proveer un mecanismo por el cual pueda dar explicaciones sobre los motivos que lo llevan a tomar una determinada decisión, es decir como hizo para inferir una

solución a partir del suministro de los síntomas manifiestos por el circuito.

4- **Posibilidad de Aprendizaje**
Puede ocurrir que la información suministrada en el momento del diseño, acerca del dominio en el que se debe manejar sea errónea ó incompleta, lo que será detectado cuando pretenda localizar una falla y no pueda lograrlo. En base a esta posibilidad deseamos que el Sistema pueda actualizar su conocimiento en forma permanente (Conocimiento Incremental).

5- **Adaptabilidad a diferentes circuitos**
Pretendemos dotar a SEDIF de una estructura que posibilite su aplicación sobre diversos prototipos, sin orientarlo a un circuito en particular. Esto implica que la misma deberá ser de características modulares, de modo que reemplazando un módulo de información por otro, pueda utilizarse el mismo Sistema Ejecutivo.

En general, lo que estamos requiriendo del Sistema Experto para el Diagnóstico de Fallas es que se comporte en una forma aproximada a la que manifiesta un Especislista Humano en Fallas.

Para otorgarle al Sistema los cinco atributos señalados, creamos tres módulos individuales que constituyen el Sistema Ejecutivo de SEDIF.

a) **Módulo de Diagnóstico.**

b) **Módulo de Enseñanza.**

c) **Módulo de Aprendizaje.**

La Figura 1 muestra el comportamiento elemental de cada Módulo.

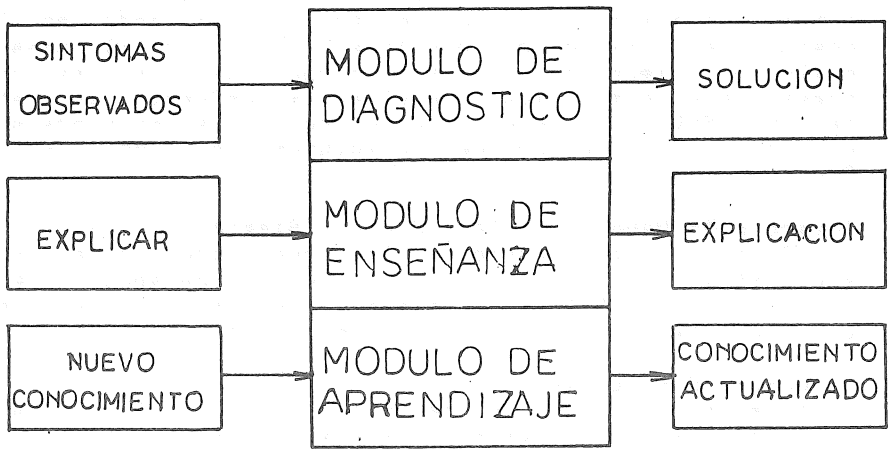


Figura 1 SISTEMA EJECUTIVO DEL SEDIF.

QUE SON LOS SISTEMAS EXPERTOS ?

Formalmente podríamos aceptar la siguiente definición de Sistema Experto:

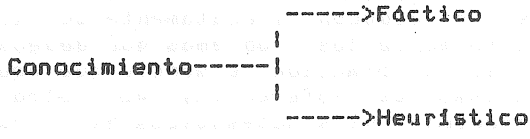
> "Se dice que un Programa de Cómputos es un Sistema Experto si al realizar una determinada tarea compleja , lo hace con tanta capacidad y eficiencia que nos parecería resuelto por un Experto Humano en esa materia."

Si analizamos rapidamente las cualidades pretendidas para SEDIF y observamos esta última definición, inferimos que el mismo no va a ser otra cosa que un Sistema Experto (en este caso el dominio de especialización será el Diagnóstico de Fallas en Circuitos Electrónicos).

Ahora bien, para simular el comportamiento de un especialista debemos hallar un modelo, aunque no sea más que una aproximación, de la estructura cognitiva del mismo, y a partir de este podremos visualizar como realiza sus actividades mentales. Escogido el modelo, el próximo paso consistirá en determinar cuales serán las herramientas computables más confiables para implementarlo. Pasemos a resolver el primer paso del problema planteado.

MODELO DEL ESPECIALISTA

Podemos realizar una división general de los Tipos de Conocimientos que normalmente maneja cualquier experto, tal como lo expresa el siguiente esquema:



a) **Conocimiento Fáctico**
Es llamado habitualmante "Conocimiento Enciclopedista" ó de "libro de texto", necesita poca elaboración puesto que es el más obvio. Se puede expresar y representar con facilidad en una Computadora.
E.j. "Si la impedancia que existe entre el colector y emisor de un transistor es nula, entonces no funcionará correctamente."

b) **Conocimiento Heurístico**
Es más difícil de archivar en una Computadora, es la red de intuiciones, reglas de juicio, "teorías de entrecasa", experiencia y en general procedimientos de inferencia que en combinación con el Conocimiento Fáctico sobre un determinado tema, le permite al hombre exhibir comportamiento inteligente.
La heurística es una propiedad intrínseca que posibilita dirigir el pensamiento a lo largo de las rutas que más verosimilmente conducen a la meta, dejando sin explorar (en los mejores casos) las avenidas menos prometedoras cuando analiza situaciones donde el número de

alternativas a considerar para llegar a la solución es significativo. Cualquier Sistema que se intente sea un recurso sofisticado de Información Inteligente debe, de algún modo, incorporar este nivel superior de conocimiento.

Analicemos como razona un especialista, ó sea que métodos emplea para inferir información nueva a partir de la existente. Tambien los tipos de razonamiento en términos generales pueden ser dos (2) tal como se expresa a continuación:

```
                          -----> Tentativo
                         |
Razonamiento-------------|
                         |
                          -----> Irrevocable
```

a) **Razonamiento Tentativo ó no Monótono**
Es el tipo de razonamiento que más comunmente emplea un especialista al postular afirmaciones plausibles en algún momento, pudiendo retractarse de ellas regresando al punto de decisión original. Esto ocurre generalmente cuando toda la información disponible en un dado instante no es suficiente para inferir nuevos resultados con absoluta certeza.

b) **Razonamiento Irrevocable ó Monótono**
Tiene que ver con el razonamiento matemático formal, los resultados se deducen direcatamente de la información previa; la forma de los problemas que deseamos abordar no presentan la información de manera que podamos escoger este mecanismo de inferencia, por otra parte donde existen procedimientos matemáticos bien definidos para resolver un problema los Sistemas Expertos son innecesarios.

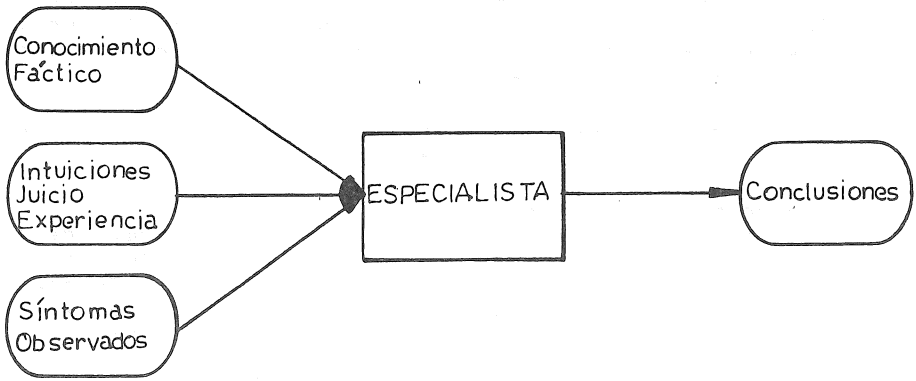


Figura 2 MODELO DEL ESPECIALISTA

La Figura 2 muestra el modelo escogido para representar globalmente la estructura cognitiva de un especialista en fallas,

quien a través de su Conocimiento Fáctico y Heurístico, de los Síntomas observados y con un mecanismo de inferencia adecuado, alcanza las conclusiones.

Un experto en fallas básicamente intentará reconocer las causas que originan el mal funcionamiento del Sistema Electrónico bajo análisis partiendo de las evidencias observadas, conociendo a priori cual es el funcionamiento correcto, aplicando eventualmente los criterios derivados del Conocimiento Heurístico.

REPRESENTACION DEL CONOCIMIENTO. RESOLUCION DE PROBLEMAS.

La situación planteada puede representarse como el intento de arribar a un **Estado Objetivo**, que es la solución del problema, a partir de un **Estado Inicial** al que podemos asociar con las evidencias manifiestas por el circuito (síntomas).

Consideremos que a partir del estado inicial pueden generarse nuevos estados aplicando las reglas de inferencia lícitas definidas por el problema; representémoslo mediante un estructura tipo árbol, donde se asocia a cada estado con un nodo del árbol y a las transiciones entre estados con las ramas del mismo tal como lo indica la Figura 3. (hemos escogido para SEDIF una de las estructuras posibles para representar el Conocimiento Fáctico, suelen ser útiles tambien los Frames y las Redes Semánticas.)

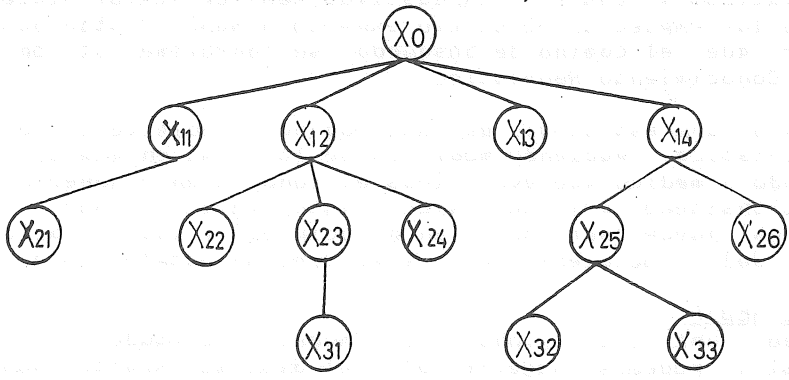


Figura 3 REPRESENTACION DEL CONOCIMIENTO FACTICO

El nodo indicado con X0 es llamado raíz del árbol y pertenece al nivel de decisión cero (0). A partir de este, se derivan X11, X12, X13 y X14 pertenecientes al nivel uno (1) y así, en forma sucesiva los nodos Xnm pertenecientes al nivel N. Tambien suele denominarse a la estructura anterior, "Espacio de Búsqueda" a partir de X0 aplicando la totalidad de las reglas permitidas. En este ejemplo; X13, X21, X22, X24, X26, X31, X32, y X33 son **nodos terminales**, o sea, aquellos en los cuales no pueden derivarse otros nuevos.

Se denomina **Estrategia de Búsqueda** a las técnicas empleadas por un especialista cuando, a partir de una situación inicial,

intenta arribar a la solución de un problema planteado. Este proceso tiene que ver directamente con la forma en que se irá moviendo a lo largo del árbol durante la búsqueda. Dada la naturaleza de los problemas que deberá resolver SEDIF, donde el estado inicial se conoce pero no así el estado final (solución), un mecanismo de preguntas y respuestas será el responsable de orientar al usuario hacia la meta. En general existe más de una alternativa para dirigirse a la solución a partir de los síntomas observados, en otras palabras hay varios caminos entre el nodo inicial y el nodo objetivo.

El **Camino Optimo** entre un nodo inicial y uno objetivo es aquel que contiene la menor cantidad de nodos intermedios, es el camino más directo ente los mismos.

En un espacio de búsqueda existen nodos que pertenecen al camino óptimo y nodos que no pertenecen; el propósito de las técnicas de búsqueda razonables es minimizar el número de este tipo de nodos a explorar para reducir al máximo el gasto empleado en arribar a la solución.

Existen técnicas o estrategias de búsqueda **exhaustivas** y no **exhaustivas**; las primeras, en general no apelan al conocimiento heurístico y se los denomina "Metodos de Búsqueda Ciega" ó "Deterministicos". Las no exhaustivas son de mayor interés, puesto que las emplea un verdadero experto humano; están basados en lograr que el camino de búsqueda se aproxime al óptimo, empleando Conocimiento Heurístico.

En general una estrategia de búsqueda no exhaustiva es de tipo no determinística, pudiendo modificarse la forma en que un árbol es explorado a medida que evoluciona el conocimiento general -el camino transitado por un especialista para arribar a sus conclusiones puede ser uno en un principio, y luego gracias al incremento del conocimiento (Ej. experiencia) adoptar otro-.

MODELO DEL SEDIF

Teniendo una idea global del modelo adoptado para el especialista, podemos sugerir el del Sistema Experto que se comporta en forma aproximada a como lo hace aquél. En ese sentido nos hemos apoyado en los denominados **Sistemas Basados en Reglas** quienes separan los dos componentes principales de la inteligencia: Razonamiento y Conocimiento. En principio esta no es una división rígida, puesto que los procesos de razonamiento se apoyan en la mayoría de los casos en el conocimiento, aunque aquí tenemos por un lado al mecanismo general de razonamiento ó motor de inferencias, tambien llamado el **Intérprete** del Sistema Inteligente. Por otro lado estará el conocimiento fáctico, conformando la **Base de Conocimiento**, la red de reglas que describen (en este caso) el funcionamiento del Circuito bajo Diagnóstico.

El intérprete manipulará el Conocimiento Heurístico, y será quien fije la estrategia de búsqueda a emplear.

La Figura 4 describe el modelo propuesto para SEDIF, quien en el modo de consulta recogerá los síntomas observados del circuito bajo ensayo a través del usuario y comenzará la búsqueda controlada por el mecanismo general de razonamiento apelando a los tipos de conocimiento vistos.

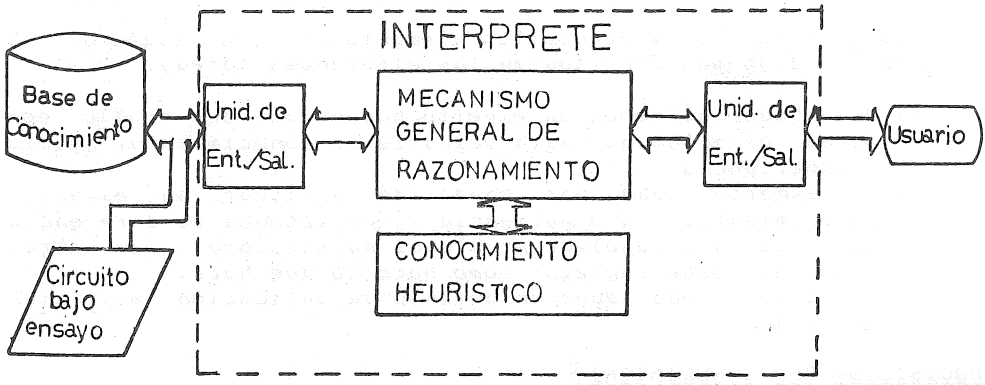


Figura 4 MODELO DEL SEDIF

Las unidades de Entrada-Salida del intérprete le permiten comunicarse con los componentes internos del SEDIF, así como con el operador.

Analicemos las ventajas de este modelo frente al de los programas tradicionales. La principal diferencia (que ya podemos visualizar) radica en la clara separación ente el Conocimiento Específico y el Mecanismo General de Razonamiento. Esta división, junto a la partición adicional del conocimiento general en reglas separadas, ofrece ventajas significativas que pasamos a enumerar:

1) Permite que la Base de Conocimiento pueda desarrollarse incrementalmente a lo largo del tiempo mediante un mejoramiento de las reglas presentes y el agregado de otras nuevas.
2) Un mismo Intérprete puede ser usado para aplicaciones diversas, mediante el recurso de cambiar un "juego" de reglas por otro.
3) Ofrece la posibilidad de que un mismo "paquete" de conocimiento pueda ser utilizado de modos diversos (incluyendo aplicaciones pedagógicas en enseñanza) adecuando el diseño del Sistema Experto. Digamos que esto último puede ser logrado con relativa facilidad tal como ocurre en el desarrollo del SEDIF.
4) El programa puede dar claras y simples explicaciones de su comportamiento describiendo las reglas aplicadas en la última sesión. Este recurso resulta de gran utilidad para encontrar reglas erróneas.
5) La posibilidad de desarrollar Sistemas que sean Instrospectivos (que puedan, por ejemplo, chequear la consisitencia de sus propias reglas para evitar contradicciones), y Evolutivos (que puedan modificar

automaticamente sus reglas, y aún aprender otras nuevas).

La **Modularidad** ofrecida por el modelo propuesto permite realizar con muy poco esfuerzo, el mantenimiento y actualización del Sistema.

Podemos citar tres requisitos elementales para asegurar el éxito de los Sistemas Expertos en las diferentes tareas:

1) Debe haber al menos un experto humano que desarrolle esa tarea correctamente para volcarle su conocimiento, juicio y experiencia.
2) El experto debe ser capáz de explicar su especial conocimiento, su experiencia y los métodos de inferencia que usa para resolver problemas particulares, en otras palabras, debe expresar como hace lo que hace.
3) La tarea debe tener un dominio de aplicación muy bien definido.

ADQUISICION DEL CONOCIMIENTO

Uno de los "cuellos de botella" que afronta la construcción de Sistemas Expertos es el elevado tiempo que se necesita para interrogar a los especialistas y representar su conocimiento especial en forma de reglas (como ocurre en nuestro caso). Este es el problema de la adquisición del conocimiento; más precisamente, la Base de Conocimiento se construye usando los mecanismos provistos por la **Ingeniería del Conocimiento**, basado en el diálogo entre el profesional y el experto en un dado campo. Las etapas involucradas en este proceso son dos (2).

ETAPA 1

a) Identificación de los conceptos y características claves del problema.
b) Conceptualización: clasificación de los conceptos y sus relaciones con el terreno del conocimiento.

ETAPA 2

c) Formalización: elección de una estructura adecuada para representar el conocimiento y el método de inferencia a ser usado.
d) Implementación: Formalización de todas las reglas específicas y de la heurística que conduce a la solución del problema.
e) Validación de las reglas y heurísticos a ser implementados.

A pesar de los esfuerzos realizados, la implementación de la adquisición del conocimiento sigue siendo uno de los problemas más difíciles. El desarrollo de un Sistema que contenga algunos pocos cientos de reglas suele llevar varios meses de trabajo a los expertos, y aún más al constructor del programa.

ANALISIS DEL CIRCUITO BAJO ENSAYO

A continuación describiremos el comportamiento del circuito que se diseñó a fin de ser utilizado como ejemplo de test para

analizar la performace de SEDIF.

Lo denominamos Controlador de Acceso a un Recinto (CAR), y su función es custodiar el ingreso en un sitio cualquiera que deba estar protegido, permitiendo solo el acceso a aquellas personas que dispongan una tarjeta con un código autorizado, e impidiéndolo a las que no lo posean. Además el CAR brinda la posibilidad de grabar ó borrar códigos que se pretendan memorizar ó eliminar respectivamente.

CAR consta de tres (3) entradas:

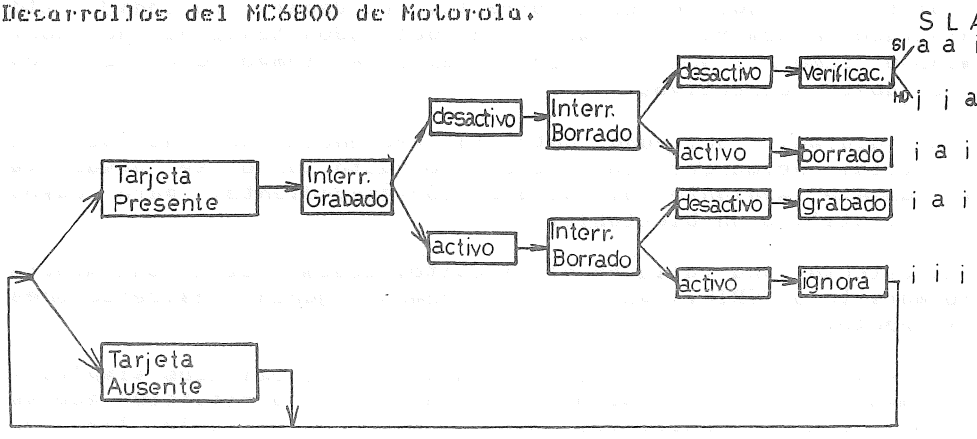
Unidad lectora de códigos.
Pulsador de borrado.
Pulsador de apagado.

y de tres (3) salidas:

Indicador luminoso.
Solenoide electromagnético.
Alarma.

En la Figura 5 podemos visualizar esquematicamente el comportamiento de este dispositivo. Observamos que si algún intruso pretende ingresar con un código no registrado en la memoria, el CAR no la verificará accionando la alarma. Cabe destacar que si por error se accionaran simultaneamente los interruptores de grabado y borrado, el Sistema ignorará la órden hasta que alguno sea desactivado.

A través de la Figura 6 apreciamos el esquema circuital del dispositivo. El mismo fué implementado en Base al Kit de Desarrollos del MC6800 de Motorola.



S : Solenoide
L : Luz
AL: Alarma

a: activo
i: inactivo

Figura 5 DESCRIPCION FUNCIONAL DEL CAR

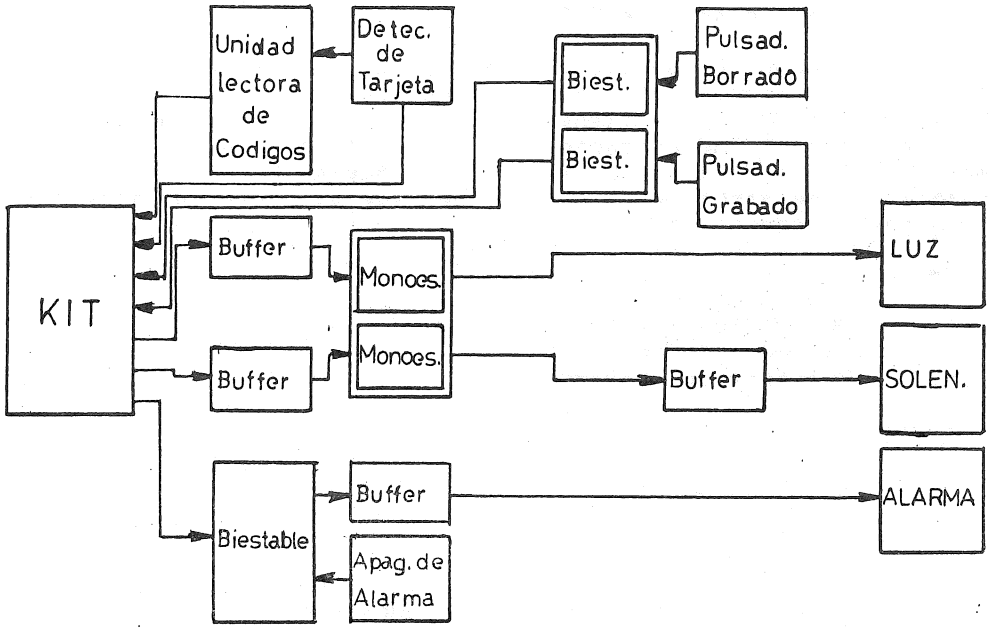


Figura 6 ANALISIS DEL CIRCUITO EN BLOQUES

Cuando se introduce una tarjeta, su presencia es acusada por un detector, el cual se encarga de informarle esta situación al Microprocesador y energizar la Unidad lectora de códigos. Los dos pulsadores envían información a través de sendos biestables que evitan aleatorios por rebotes en los contactos de los mismos. Si la tarjeta introducida fuera correcta, es decir, su código está registrado en memoria, permanecerán activados tanto el indicador luminoso como el solenide por un lapso de tiempo dado por los monoestables respectivos.

La alarma, activada ante la presencia de una tarjeta incorrecta, permanece en ese estado hasta tanto se desactive externamente mediante el pulsador correspondiente, para cuyos fines se utilizó un biestable.

Al detectar una falla en el circuito, debemos de alguna manera informárselo a SEDIF, por lo que haremos algunas consideraciones al respecto.

En general habrá dos tipos de tests, los denominados simples y los compuestos. Los primeros están dados por la situación que se presenta cuando analizamos las salidas con un solo tipo de tarjeta (correctas e incorectas en forma separada). Por ejemplo, verificar el estado de la alarma con una tarjeta correcta. Los tests compuestos se conforman analizando tanto la situación de las salidas para una tarjeta correcta como para una tarjeta incorrecta.

Cada una de esas situaciones representará un síntoma posible, los que a su vez se convertirán en los nodos iniciales ó raices de los "subárboles" que conforman la Base de Conocimiento.

REPRESENTACION DEL CONOCIMIENTO

Veamos sintéticamente como representar una "porción" de conocimiento del CAR. Como se observa en la Figura 7, es una Estructura tipo árbol donde un síntoma está expresado en términos de la raiz, y cuyos nodos teminales serán los posibles resultados del mal funcionamiento del Circuito que manifiesta ese síntoma. Si uno lo desea, puede seguir derivando estos últimos hasta resolver al nivel de definición que se pretenda (bloque ó componenete). En pariculár, el árbol tomado a título de ejemplo representa lo que acontece en un bloque de la interfase de salida pero no resuelve un inconveniente que pueda presentarse en el Kit, indicado en este caso como el bloque "logica", lo que no significa que el SEDIF no puede estar capacitado para diagnosticar inconvenientes en el mismo; todo lo contrario, el usuario puede "colgar" nuevos subárboles ó ramas del nodo en cuestión y resolver fallas en los componentes del Kit. Esto es importantísimo ya que le permite al SEDIF poseer una propiedad natural del hombre , el **Conocimiento Incremental**, adquiriendo más información sin eliminar la existente.

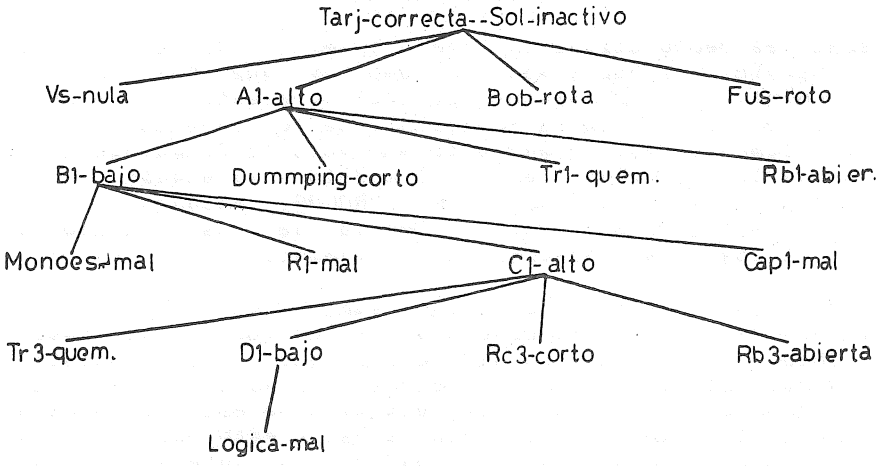


Figura 7 ESPACIO DE BUSQUEDA

Los nodos del árbol en este caso son **Nodos OR**, es decir que existe independencia entre los estados de un mismo nivel de decisión, lo que en algunas ocasiones puede no ser así. En estos casos la solución está en incorporar a ese nivel nuevas ramas que contemplen tales posibilidades. La Base de Conocimineto del SEDIF contiene 69 Reglas.

LA NECESIDAD DE LOS LENGUAJES DECLARATIVOS

Escoger el lenguaje que permita implementar cualquier proceso en un Computadora, significa fundamentalmente analizar la

adaptabilidad de las Estructuras de Control y de Datos, ofrecidas por las versiones disponibles, para ejecutar y representar dicho proceso. En este caso buscamos codificar el modelo propuesto para el Sistema Experto representado en la Figura 4.

Si analizamos las estructuras de control de los diferentes lenguajes, nos encontraremos con dos grupos generales bien definidos. Por un lado los que se basan en el modelo de la máquina propuesta por Von Neumman, donde un programa es una colección de órdenes ó instrucciones a ejecutar en forma secuencial, salvo excepciones, tal como lo hace la Unidad Central de Procesamiento quien ejecuta el trabajo al más bajo nivel. Si bien, gracias a los compiladores la programación en lenguajes de alto nivel no se torna tediosa, desde el punto de vista de la "abstracción" no hemos avanzado demasiado ya que permanece intacta la idea "procedimental" de Von Neumman; por esta razón a los lenguajes que se ajustan a este modelo rígido se los denomina **Lenguajes de Procedimiento** (Procedural Languages) ó **Imperativos**. BASIC y FORTRAN integran este grupo. Al ser la Estructura de Control bastante rígida, vinculada a la forma en que operan las partes de una Computadora, cuando queremos codificar procesos que no se ajustan a este modelo, ya que se hace necesario plantear uno abstracto, (como ocurre con los procedimientos inteligentes) resulta muy difícil ó casi imposible hacerlo.

Los **Lenguajes Declarativos** en cambio intentan "declarar" que es lo que se pretende hacer más que como hay que hacerlo, la estructura semántica de los mismos permite representar un modelo abstracto (en el sentido de la relación ente los objetos de ese modelo y los elementos que normalmente manipula la Computadora) y es precisamente la "interfase" entre el modelo y la máquina. Los más difundidos en este grupo son LISP y PROLOG, basados en dos disciplinas formales como son la **Teoría de las funciones recursivas** y la **Lógica matemática** respectivamente.

Cuando se analiza el tipo de datos a emplear para representar la información de cualquier procedimiento computable, se busca (en lo posible), estrechar las diferencias que puedan existir entre la forma adoptada por la representación natural (la que manipula el hombre) y la forma representativa de esa información que debe manejar la Computadora. Esto es así puesto que conviene que el programador dedique su mayor esfuerzo a interpretar el funcionamiento general del Sistema sin entrar en los detalles que no hacen la comportamiento mismo de este (no por ello menos trascendentes en el diseño de un programa); esto, naturalmente tiene mayor sentido en los problemas complejos.

Los elementos que deberá manejar SEDIF, más que números, serán símbolos, por lo que los datos tendrán que ser de características simbólicas, en particular nos interesarán aquellos que permitan representar las reglas del tipo SI-ENTONCES. Por ejemplo:

> "Si el punto A1 del circuito permanece en nivel lógico alto, ó la tensión Vs es nula, ó la bobina está rota, ó el fusible está cortado, entonces al colocar una tarjeta

correcta el solenoide no actuará."

El texto anterior expresa una porción de conocimiento de SEDIF sobre el CAR, es decir es una parte de la Base de Conocimiento. Podemos decir lo mismo de la siguiente forma:

(tarj-correcta--solenoide-inactivo (Vs-nula Bob-rota Fus-cort A1-alto))

La regla ha sido expresada en términos de una lista cuyo primer elemento (conclusión de la regla) es un átomo y el segundo (las premisas) es otra lista de átomos. (Un átomo es una cadena de símbolos alfanuméricos y/o ciertos caracteres especiales). Observamos que el implicante lógico y la disjunción lógica están implícitos en la estructura adoptada. Precisamos un formalismo que permita tratar con este tipo de datos, es decir manejar listas. LISP(LISt Programming) ofrece recursos poderosos para esto último, es el lenguaje más difundido en el ámbito de la IA y pertenece al grupo de los **Lenguajes Funcionales**, en los que la programación consiste en definir funciones. (los lenguajes funcionales integran un grupo más general, el ya comentado, los lenguajes declarativos.

Una de las razones principales que nos impulsaron a elegir LISP, se basó en el análisis de la Estructura de Datos necesaria para almacenar el conocimiento del Sistema Experto ó sea representar la Base de Conocimiento en alguna forma que se facilite su creación, acceso y mantenimiento. La versión de LISP adoptada, permite asociarle a cualquier variable una **lista de propiedades**, ó sea que cada variable podrá tener asignado un conjunto de propiedades así como una persona tiene determinados atributos que lo identifican (sexo, número de documento, edad y otros).

La forma que toma la estructura mencionada es la siguiente:

	PROPIEDAD 1	PROPIEDAD 2	- - - - - - -	PROPIEDAD N
VARIABLE	Valor1	Valor2	- - - - - - -	ValorN

	CONCLUSION	PREMISAS
REGLA 1	tarjeta-correcta--sol-inactivo	(Vs-nula A1-alto Bob-rota Fus-roto)

Figura 8 LISTAS DE PROPIEDADES

En la Figura 8 se muestra tambien, la forma adoptada para una regla, observamos que los valores que pueden tener asignados para

las diferentes propiedades serán, ó bien átomos, ó bien listas. Un atributo saliente de LISP es que absorve la recursión, una propiedad matemática que les permite a las funciones definirlas en términos de ellas mismas. Ej (potencia, factorial).

$$M^{n} = M * M^{(n-1)}$$

$$N! = N * (N - 1)!$$

La utilidad que le reporta la recursión a LISP está manifiesta en que el concepto matemático se extiende al simbólico, permitiéndole el uso de ella para hacer lo que mejor conoce LISP, manejar listas. Una persona sabrá programar en LISP cuando domine este simple y poderoso recurso.

Una gran cantidad de fenómenos naturales, incluso algunos procedimientos mentales son de naturaleza recursiva, lo que nos permite codificarlos sin necesidad de simularlos. En conclusión, analizando la Estructura de Control y de Datos, el manejo simbólico y la recursividad que ofrece LISP, decimos que este es un lenguaje adecuado para implementar el Sistema de las características ya señaladas.

DESCRIPCION FUNCIONAL DEL INTERPRETE

Al observar un funcionamiento incorrecto en el circuito llamamos a SEDIF y le indicamos cuáles son los síntomas observados. Por ejemplo.

```
tarj-correcta--sol-inactivo
```

A partir de este punto comienza a explorar el árbol de la Figura 7, y el objetivo es verificar si el problema está en alguno de los elementos de la lista **objetivos**, que en este punto valdrá:

```
objetivos = (A1-alto Vs-nula Bob-rota Fus-roto)
```

Digamos que la forma en que el intérprete explora el árbol dependerá de los **factores de decisión** asociados a cada nodo, teniendo mayor prioridad los de valores numéricos superiores. Este factor de decisión (D), depende directamente del número de veces que exploró dicho nodo en el pasado (se pondera la experiencia). Por otro lado será inversamente proporcional al nivel de profundidad (P) asociado a dicho nodo, el que se toma como el número de nodos, a partir de él, a recorrer para llegar al nodo terminal más alejado. Un nodo será el terminal más alejado de otro, si el primero cumple con estas condiciones: ó es el segundo tomado como referencia, ó se deriva del segundo, ó se deriva de otro que se deriva del segundo (notar el procedimiento recursivo de la última condición).

El último criterio permite realizar eventualmente la " vuelta hacia atrás" más corta cuando el camino escogido no conduce a la solución; es el "por las dudas". Veamos los niveles de profundidad asociados a cada elemento de la lista objetivos.

Nodo	Profundidad (P)
A1-alto	5
Vs-nula	1
Bob-rota	1
Fus-roto	1

Si pensamos que el número de veces que se transitó por esos nodos en el pasado es el mismo, convendrá escoger un terminal y nó el no terminal ya que si así no fuera (y la solución fuese alguno de los terminales), deberíamos volver hacia atrás con el consiguiente costo asociado. En esta situación SEDIF elegirá uno terminal; cuando los factores de decisión de dos ó más nodos sean iguales tendrán mayor prioridad los que se encuentren mas a la izquierda en la lista objetivos.

El criterio de búsqueda adoptado, si bien está relacionado con las ideas generales que emplea un experto, no deja de ser empírico; la forma de validar a los heurísticos es mediante la prueba misma del Sistema analizando si los resultados arrojados son satifactorios, obviamente no existe un "paquete estandarizado" de heurísticos que se adapte a cualquier clase de problemas.

El hecho de que la estrategia escogida sea tentativa implica la necesidad de dotar al intérprete de un mecanismo apropiado para regresar al punto original si la solución no está en el camino elegido en primer término, dicho mecanismo es el **BackTracKing** y es muy común en los Sistemas basados en IA. El diseño del intérprete consistió en definir 40 funciones relacionadas como lo muestra la figura 9.

El intéprete además tiene en cuenta situaciones particulares como ser los problemas en el correcto suministro de energía, ó de interpretación errónea por parte del usuario del tipo de tarjeta que está utilizando el el test; en cuyos casos intenta en primer término verificar este tipo de problemas graves ó groseros y que están más a la vista. Esto se condice con el comportamiento del especialista cuando dá un "vistazo" al circuito bajo análisis, para verificar en función de ciertas evidencias si el problema puede detectarse a simple vista, antes de comenzar la búsqueda "fina" ó más detallada.

Volviendo al principio de este apartado, cuando el intérprete encuentra que el nodo cuyo factor de decisión más grande corresponde a uno terminal, busca el resto de los nodos terminales que pertenecen al mismo nivel preguntando (en el caso de que queden en la lista objetivos otros elementos) si alguno de ellos es la solución ó infiriéndola en el caso de que sean los únicos elementos de objetivos. Supongamos que el nodo de mayor factor de decisión es A1-alto, entonces debe seguir explorando el árbol hacia abajo derivando nuevos nodos, quedando objetivos de la siguiente forma:

```
objetivos = (Tr1-quem Dummping-corto B1-bajo Rb1-abier (Vs-nula
                Bob-rota Fus-roto))
```

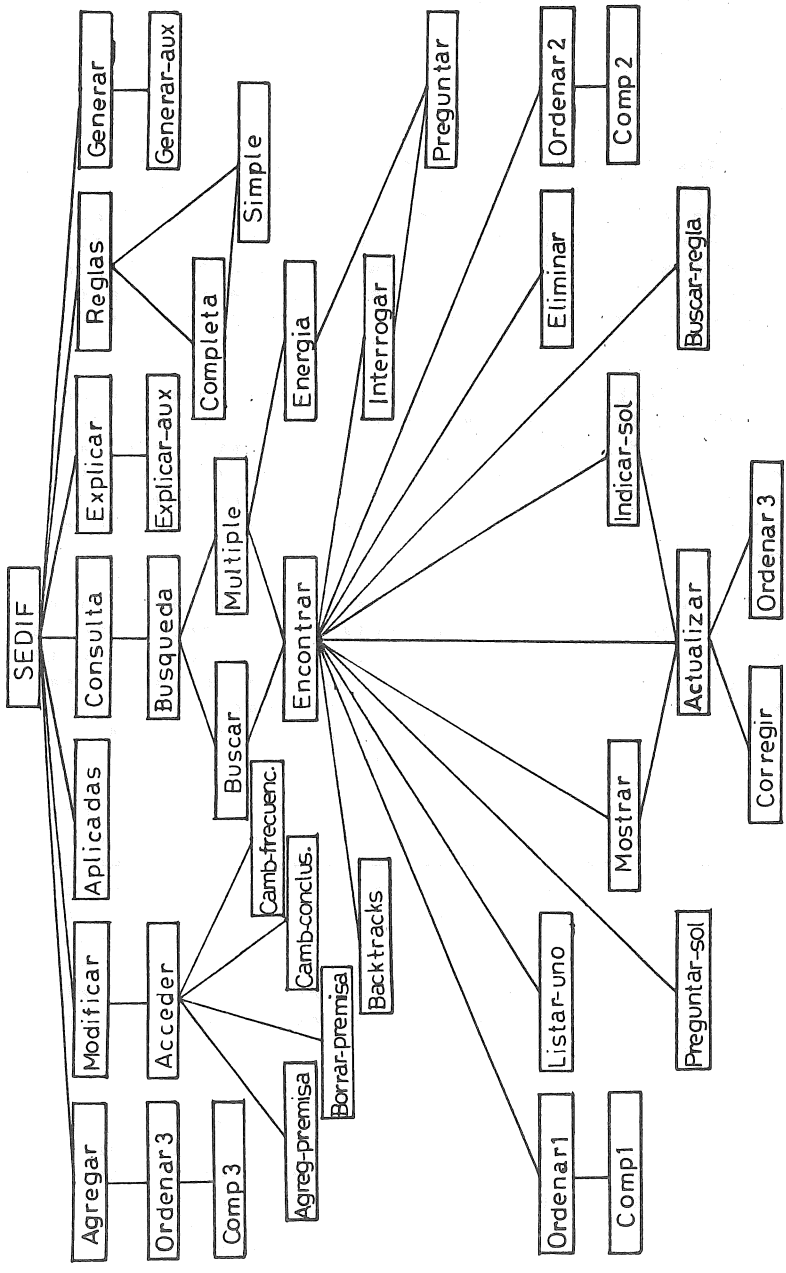


Figura 9 FUNCIONES DEL SEDIF

Los nodos que serán analizados en última instancia quedarán en las posiciones más profundas de la lista. Si el mayor factor de decisión fuese ahora (para los nodos menos profundos) el correspondiente a Tr1-quem el intérprete preguntará:

(VERFIQUE SI ALGUNA DE ESTAS SITUACIONES SE PRESENTA "SI/NO")
(TR1-QUEM DUMMPING-CORTO RB1-ABIER)

A continuación, el usuario, haciendo uso de los elementos apropiados (tester, punta lógica, osciloscopio, etc.) podrá responder al interrogante planteado. Supongamos que la respuesta sea NO, luego objetivos pasa a valer:

objetivos = (C1-alto Monoes-mal R1-o-Cap1-mal (() (Vs-nula Bob-rota Fus-roto))

La presencia de la lista vacía en el segundo nivel, expresa que en el mismo nivel del árbol no quedan alternativas por explorar. El proceso de agregar nuevos elementos en la lista objetivos continúa hasta que, ó bien el usuario responde afirmativamente una pregunta indicando cual, de la lista que eventualmente indique SEDIF, es la solución del problema, ó bien llega a un nodo de profundidad 2 (D1-bajo) donde comienza el backtracking si las consultas que realiza el Sistema son respondidas negativamente. Así como la expansión hacia abajo consistía en agregar términos en la cabeza de la lista objetivos, el backtracking o retroceso hacia arriba consistirá en eliminar dichos elementos comenzando con los ubicados en los niveles más bajos de la lista en cuestión. El proceso de Backtracking no continúa necesariamente hasta la raíz, pudiendo recomenzar la búsqueda abajo (escogiendo otro camino) en cualquier nivel del árbol. La búsqueda hacia abajo, se realiza bajo el control de la función encontrar (el corazón del intérprete); mientras que el retroceso lo efectúa a través de backtracks.

Una vez que SEDIF resuelve un problema alcanzando a un nodo terminal que es la solución, comienza, antes de salir, un proceso de actualización incrementando en uno (1) a las variables (que indican el número de veces que el intérprete pasó por allí) de los nodos pertenecientes al Camino Optimo actualizando los factores de decisión de dichos nodos. Ademas realiza lo propio con los valores de la propiedad frecuencia de las reglas aplicadas, permitiendo que en las sucesivas sesiones sean consideradas con mayor prioridad que antes.

Con estas ideas explicamos brevemente cual es el espíritu de la búsqueda del SEDIF. Puede llegar a ocurrir que se explore todo el árbol y la solución no sea encontrada, en otras palabras que la lista objetivos sea finalmente la lista vacía. En tales casos el Sistema Experto le comunica al usuario que con la información actual no puede diagnosticar la falla, debiendo averiguarlo por otros medios y cuando lo haya conseguido puede invocar a SEDIF informándolo sobre las razones que provocan el mal funcionamiento del circuito, aprovechando la facilidad de mantener su Base de Conocimiento.

Cuando el Sistema infiere que la solución está entre un grupo de posibles causas y no puede distinguirlas le responde con una lista de ellas de izquierda a derecha según los valores de certeza decrecientes.

CONCLUSIONES

Hemos logrado que SEDIF manifieste las características apuntadas al principio de este trabajo, mediante el empleo de recursos de programación no convencionales que van desde el modelo en sí mismo hasta el propio lenguaje en el que fué implementado SEDIF. En conclusión, el Sistema Experto para el Diagnóstico de Fallas:

- posee un razonamiento no determinístico, modificando el sentido de la búsqueda según los resultados del pasado. Pudimos observar que al principio realiza un número significativo de preguntas, el que disminuye a medida que aumenta el número de consultas. Esto último es exactamente lo que acontece con un especialista que comienza a resolver un nuevo problema en su campo.
- controla la búsqueda y orienta al usuario asistiéndolo en forma permanente.
- puede explicar como hizo para llegar a la solución a partir de los datos iniciales suministrados por el usuario.
- puede actualizar su conocimiento con facilidad gracias a la estructura de datos (listas de propiedades) ofrecida por LISP.
- se adapta a diversos circuitos debido a que el modelo propuesto separa los componentes principales de la inteligencia: el razonamiento y el conocimiento.

Los Sistemas Expertos están en una etapa de crecimiento continuo y van avanzando progresivamente en el ámbito de la Informática y de las otras disciplinas. Surgiendo de la Inteligencia Artificial, y siendo cada día más poderosos, los Sistemas Expertos realizan un aporte valioso en la mayoría de las Ciencias, no por remplazar a los especialistas, sino por ser una herramienta complementaria de alto valor científico que los asiste en su campo.

BIBLIOGRAFIA

Nillson, N. :"Principles of Artificial Intelligence"; Springer Verlag (1982).

Winston, P. :"Artificial Intelligence"; Mc Graw Hill (1978).

Hassemmer, T. :"Looking at LISP"; Microcomputer Books (1983).

Furtado, A :"Paradigmas de Linguagens de Programaçao"; IEABI Campinas (Brasil) 1986.

Piaget, J :"Sicologia de la Inteligencia"; Psique (1979).

ROSARIO, Julio de 1986.

O CONCEITO DE ENTIDADE NA MODELAGEM SEMANTICA DE DADOS

José Palazzo Moreira de Oliveira
José Mauro Volkmer de Castilho
Clésio Saraiva dos Santos
Universidade Federal do Rio Grande do Sul
Porto Alegre - *Brasil*

Resumo

O desenvolvimento de sistemas de informação, apoiados por bancos de dados, exigiu o desenvolvimento de modelagens complexas da realidade. Os sistemas de gerência de banco de dados oferecem modelos de dados que permitem representar os objetos relevantes para uma aplicação. Os diversos modelos de dados apresentam um conjunto variado de conceitos o que leva a dificuldades na seleção das alternativas possíveis para a modelagem. Neste artigo é apresentado um modelo, o **modelo E**, que pelo emprego de um único conceito de base, **a entidade** e de operadores de tipo, permite uma representação natural e ao mesmo tempo completa da realidade modelada.

Palavras-chave

Modelo de dados, banco de dados, modelagem de sistemas de informação.

1. Introdução

A modelagem de sistemas de informação (SI) constitui-se em um dos problemas de solução mais complexa na área de processamento eletrônico de dados (PED). Isto é decorrência de ser a modelagem uma atividade interdisciplinar. E' necessário o trabalho coordenado de especialistas em computação com os usuários. Os primeiros conhecem as soluções técnicas disponíveis e os segundos tem o conhecimento das necessidades das aplicações. O problema da comunicação é grave. Diversas técnicas foram desenvolvidas na área de análise de sistemas para procurar resolver a dificuldade de comunicação [Mar 78].

Em PED a modelagem dos sistemas de informação está muito ligada ao modelo de dados utilizado pelo sistema de gerência de banco de dados (SGBD) escolhido. Um dos itens mais importantes para permitir uma boa comunicação entre os usuários e os técnicos

em BD é a adequação dos conceitos existentes no modelo de dados aos objetos modelados. Neste artigo é apresentado um modelo útil na descrição de sistemas de informação e facilmente utilizavel para a implantação de um SGBD.

Os diversos modelos de dados disponiveis apresentam uma grande variedade de conceitos tais como: relações, atributos, domínios [Cod 70], entidades, relacionamentos, valores [Che 76], registros, campos e "cosets" [CODASYL 71], entre outros. Uma análise apressada poderia levar á conclusão , errônea, de que quanto maior o número de conceitos disponiveis em um modelo mais facilidade o mesmo oferece para a modelagem.

Entretanto, devido á interdependência entre os conceitos pode levar o usuário a sérias dificuldades para a classificação dos objetos a serem modelados em razão da subjetividade que passa a caracterizar este processo.

Pode ser citado como exemplo o Modelo Entidade-Relacionamento (ER), que não apresenta uma perfeita distinção entre os conceitos de Entidade, Relacionamento e Valor. Em um banco de dados de ensino, **turmas** podem ser descritas como entidades, tendo **professor**, **disciplina** e **sala** como atributos, ou alternativamente, **turma** pode ser modelada como um relacionamento entre as entidades **professor**, **disciplina** e **sala**.

2. Os modelos semânticos

Um modelo de dados é um formalismo utilizado para representar os objetos que pertencem á parte da realidade correspondente a um grupo de aplicações. A semântica associada aos objetos corresponde ás propriedades de estrutura e de comportamento dos objetos selecionados. Ela leva em consideração, em particular, o efeito dos operadores existentes sobre os objetos. Por outro lado torna-se cada vez mais importante a possibilidade de uma representação mais fiel da realidade. Esta necessidade tem sido detetada e diversos trabalhos propõem modelos ditos **semânticos** [Cod 79], [HM 81].

Estes modelos procuram incorporar uma maior parcela da semântica da realidade no modelo de dados através de novos tipos e de operadores de tipo. Entretanto a complexidade do modelo aumenta cosideravelmente. Em particular a extensão do modelo de dados relacional [Cod 70] apresentada por Codd no modelo RM/T [Cod 79] apresenta um sem número de conceitos (entidades do núcleo, entidades caracteristicas, entidades associativas, entidades do núcleo interno, "nonentity associations", generalização, agregação, "cover aggregation", generalização alternativa) que tornam seu emprego para a modelagem extremamente complexo.

A partir destas constatações, no presente artigo, é proposto

um modelo de **Entidades** (modelo **E**) baseado no conceito de **entidade** e nos construtores de tipo **soma, especialização, produto** e **correspondência**. O objetivo deste modelo é simplificar o processo de modelagem pelo emprego do conceito único de entidade. Desta forma é eliminada a necessidade de enquadrar um objeto em uma de várias classes alternativas. Por outro lado, os construtores de tipo proporcionam a possibilidade de combinação de entidades para a obtenção de tipos mais complexos e ajustados aos objetos que devem ser modelados. Pela simplicidade e clareza de sua definição, o modelo E apresenta uma definição formal simples e direta. Além disto, de um ponto de vista prático, poupa o usuário da desconfortavel tarefa de descrever suas aplicações com alta dose de subjetividade e com um número de alternativas que cresce exponencialmente com o número de objetos a serem modelados.

3. O modelo E

O modelo E está baseado no modelo E-R estendido proposto em [SNF 80] como uma evolução do modelo Entidade-Relacionamento (ER) [Che 76]. O modelo **E** considera como inicial o conceito de entidade. Uma entidade corresponde a um objeto, fato ou evento sobre o qual devem ser mantidas informações.

Entidades podem compor outras entidades. A forma de caracterizar o detalhamento, ou composição, de entidades no modelo **E** é a definição de entidades por meio de construtores de tipo. Um construtor de tipo combina tipos de entidade já definidas para criar novos tipos de entidade. Como o resultado de um operador de tipo é um novo tipo de entidade as estruturas de derivação são homogêneas. Isto é, os operadores de tipo podem ser aplicados sucessivamente pois o resultado será sempre uma entidade. O modelo **E** oferece quatro construtores de tipo: **especialização, soma, produto e correspondência**.

3.1 Especialização e Soma

Considerando o tipo de entidade **t**, uma especialização de **t** é um novo tipo de entidade **t'** cujas ocorrências são elementos de um subconjunto do conjunto de ocorrências de **t**. Por exemplo, considerando um tipo de entidade **funcionario**, o tipo de entidade **engenheiro** pode ser definido como:

engenheiro = especialização de (funcionário)

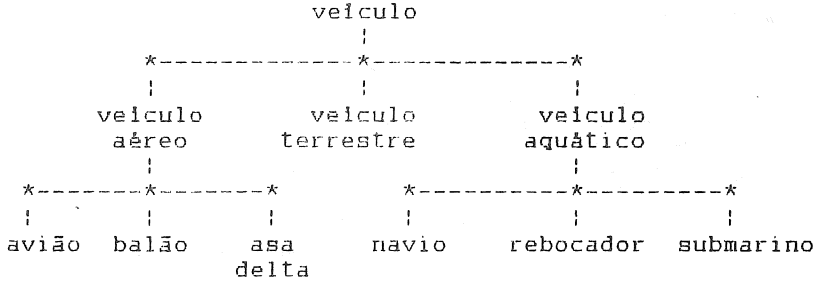
Uma observação deve ser feita aqui, em relação a este tipo de construtor. Uma entidade de tipo **engenheiro** é também uma entidade de tipo **funcionário**. Assim a criação de uma entidade do tipo **engenheiro** no banco de dados, implica na criação de uma entidade do tipo **funcionário** no mesmo banco de dados.

Se t1,t2,...,tn são tipos de entidade, a **soma** de t1,t2,...,tn é um novo tipo de entidade t cujas ocorrências são ocorrências de qualquer um dos tipos operandos. Por exemplo, considerando entidades do tipo **engenheiro** e **técnico**, o tipo de entidade **membro-de-equipe** pode ser definido como

membro-de-equipe = soma de (engenheiro, técnico)

O construtor soma é complementar ao construtor **especialização** e permite ver um tipo de objeto de nivel mais alto como a composição de tipos de objetos de nivel inferior.

O conceito de "soma" permite a obtenção de hierarquias de generalização. O exemplo citado por Smith e representado a seguir pode ser modelado com o auxílio deste construtor.

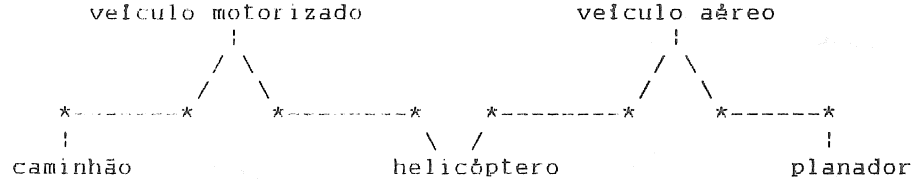


veículo aéreo = **soma** de (avião, balão, asa delta)

veículo aquático = **soma** de (navio, rebocador, submarino)

veículo = **soma** de (veículo aéreo, veículo terrestre, veículo aquático)

No caso em que é desejada a representação da participação de um tipo de entidade em duas hierarquias de generalização deve ser empregado o construtor de **soma**.



veículo motorizado = **soma** de (caminhão, helicóptero)

veículo aéreo = **soma** de (helicóptero, planador)

Correspondência

Se t é um tipo de entidade, uma correspondencia de t é um novo tipo de entidade t' cujas ocorrências são conjuntos de ocorrências de t. Por exemplo, um tipo de entidade **equipe** pode ser definida como:

equipe = correspondência de (membro-de-equipe)

A restrição de integridade associada a este construtor é:

"Uma entidade somente pode participar como componente em uma correspondência se for uma ocorrência de seu tipo."

Uma correspondência pode ser manual, quando as inserções devem ser realizadas explicitamente, ou automática. Neste último caso é especificada uma condição de pertinência que, quando satisfeita, desencadeia a inclusão automática da entidade no conjunto definido pela correspondência.

3.3 Produto

Os construtores **soma**, **especialização** e **correspondência** permitem a modelagem de interdependências entre as entidades. Esta modelagem é dependente tão somente das entidades e independe da descrição detalhada das propriedades destas entidades. Para agregar as diferentes propriedades em uma entidade é empregado o construtor **produto**. Este construtor é independente do grupo anterior e corresponde ao conceito de agregação de Smith & Smith.

Se t1 , t2,..., tn são tipos de entidade, um produto de t1,t2,...tn é um novo tipo de entidade t cujas ocorrências são n-uplas compostas com ocorrências de cada tipo participante do produto. Cada componente de uma ocorrência de t está associado ao **papel** que aquele componente desempenha no produto. A caracterização dos papeis existentes num produto pertence à definição do mesmo. Por exemplo, um tipo de entidade **departamento** pode ser definido sobre tipos **funcionário**, **equipe** e **recursos** como:

```
departamento = produto de (
   CHEFIA: funcionário,
   CORPO-TECNICO: equipe,
   INFRA-ESTRUTURA: recursos )
```

Os exemplos acima podem ter dado a impressão de que apenas objetos "concretos", como engenheiros, chefes, departamentos, são entidades. Pelo contrário, também objetos tais como uma determinada idade, ou um determinado salário podem ser encarados como entidades. Deste ponto de vista, pode ser definido um tipo

de entidade **funcionário** como o produto de tipos de entidade **nome-próprio**, **endereço**, **idade**, **salário**, como descrito a seguir:

```
funcionário = produto de (
    NOME: nome-proprio,
    RESIDENCIA: endereço,
    IDADE: idade,
    REMUNERACAO: salário )
```

A noção, geralmente difundida, de um funcionário é a de um objeto concreto ao qual estão associados valores de idade, nome-próprio, etc. Do ponto de vista da modelagem que está sendo construida, uma ocorrência de entidade **funcionario** é apenas uma ocorrência do produto definido, e não o funcionario (pessoa) em si. A associação entre um funcionário (pessoa) e a entidade do tipo **funcionário** correspondente faz parte do processo de abstração executado pelo analista durante a definição do modelo da aplicação.

Com os conceitos de **entidade** e de operadores de tipo é possivel modelar um conjunto complexo de objetos do mundo real por meio de tipos definidos de forma natural, isto é, próxima á estrutura percebida na realidade. Para isto é de grande valia a possibilidade de combinação sucessiva de operadores de tipo. A decisão fica restrita á escolha da combinação de operadores de tipo necessários para representar uma dada percepção da realidade.

O anexo 1 apresenta definições de tipos de entidades para uma aplicação relativa a projetos e departamentos.

4. Base de dados

Uma tipo **base de dados**, no modelo E, é também um tipo de entidade, mais precisamente, de uma correspondência definida sobre os tipos de entidade identificados no processo de modelagem de dados. Por exemplo, parte da definição de uma base de dados envolvendo os tipos de entidade apresentados como exemplo na secção anterior pode ser descrita como abaixo:

```
base-de-dados = correspondência de (
 soma de (
    . . .
   NOMES-PROPRIOS;
   SALARIOS;
   TAREFAS;
   FUNCIONARIO;
   DEPARTAMENTOS;
    . . .
          )          ).
```

A ocorrência do tipo de entidade **base-de-dados**, em um certo instante, é vista como o estado da base de dados naquele instante. Não está excluida a possibilidade de coexistirem várias ocorrências da base de dados, correspondendo a vários estados da base em diferentes instantes.

A interdependência entre ocorrências de tipos de entidades diferentes na base de dados é função da interdependência entre os tipos correspondentes. A associação fica estabelecida implicitamente pelo mecanismo que garante a restrição de integridade estrutural existente entre tipos e suas especializações. Assim a existência de uma entidade de tipo **engenheiro** na base de dados implica na existência de uma correspondente entidade de tipo **funcionario**.

5. Mecanismo de construção e referência a entidades

As entidades que participam de uma ocorrência ou estado da base de dados podem ser interdependentes, como mencionado anteriormente. A estrutura geral das interdependencias é vista como o esquema de caminhos de acesso ás entidades. Assim, por exemplo, uma entidade de tipo **funcionario** pode estar relacionada a uma entidade de tipo **engenheiro** (pois **engenheiro** é uma especialização de **funcionário**) e também estar relacionada com uma entidade de tipo **departamento** (é o componente com papel de CHEFIA), e indiretamente relacionada com uma entidade de tipo **projeto** (é componente com papel de LIDER no projeto).

O esquema de interdependência é montado com a definição de "operadores de referência", que apresentam ou denotam a entidade relacionada com a entidade dada. Por exemplo, seja **n** uma entidade do tipo nome-proprio. A entidade do tipo engenheiro relacionada com n seria denotada por:

engenheiro [NOME = n]

que se le como "a entidade de tipo **engenheiro**, onde n é componente com papel NOME".

5.1 Construção de ocorrências de entidades

Os operadores **Cria**, **Destroi**, **Altera**, **Insere** e **Remove** permitem a manipulação das ocorrências na base.

Cria (t, e)

cria uma ocorrência da entidade t e corpo e. O corpo é uma apresentação textual da entidade montada de forma conveniente.

Destroi (t, e)

destroi uma ocorrência **e** de entidade do tipo **t**.

Altera (t, e, p, el)

na ocorrência **e** de entidade do tipo **t**, definido como um produto, **el** passa a ser o componente com papel **p**.

Insere (t, e, el)

na ocorrência **e** da entidade do tipo **t**, definido como uma correspondência manual, insere a ocorrência de entidade **el** como elemento.

Remove (t, e, el)

na ocorrência **e** de entidade do tipo **t**, definido como uma correspondência manual, remove a ocorrência de entidade **el** do conjunto.

5.2 Operadores de referência a entidades

Considerando **e** uma entidade, **t** um tipo de entidade e **p** um papel, definimos, informalmente os cinco operadores que se seguem:

Existe (t, e)

tem o valor VERDADEIRO se existe na base de dados uma ocorrência **e** de entidade do tipo **t**.

t [p = e] (operador de seleção)

denota a entidade de tipo **t** da qual **e** é o componente com papel **p**.

componente (t, e, p)

denota a entidade que é o componente da ocorrência **e**, do tipo **t**, com papel **p**.

conjunto (t, e)

denota a entidade do tipo **t** que é uma correspondência da qual **e** é membro.

elemento (t, e)

denota uma entidade que é um elemento da entidade **e** do tipot, definido como uma correspondência.

5.3 Exemplos de referências a entidades

A seguir são apresentados alguns exemplos de aplicação dos

operadores de referência a entidades.

a) turma [código = "T500"]

denota a ocorrência de entidade do tipo turma que tem "T500" como componente com papel código.

b) componente (turma, turma [código = "T500"], professor)

denota a ocorrência da entidade que é componente com o papel **professor** da turma "T500".

c) conjunto (alunos, aluno [nome = "André])

denota a entidade do tipo alunos (correspondência) que tem o aluno André como um de seus elementos.

d) elemento (alunos, correspondência (turma, turma [código = "T500"], alunos)

denota um dos elementos (aluno) do conjunto de alunos da turma "T500".

Cada operador leva à denotação de de apenas **uma** entidade. Caso haja mais de uma candidata à denotação, uma delas será escolhida arbitrariamente. O operador **próx** aplicado aqueles operadores (seleção, conjunto e elemento) que comportam a existência de várias entidades candidatas, denota uma das candidatas ainda não excolhida desde a última aplicação do operador de referência.

e)
```
n1 <-- cria (nome_próprio, "João")
n2 <-- cria (nome_próprio, "Pedro"  )
n3 <-- cria (nome_próprio, "Paulo"  )
n  <-- cria (nomes, n1, n2, n3)
n4 <-- elemento (nomes, n)
n5 <-- próx (nomes, n)
n6 <-- próx (nomes, n)
```

Neste exemplo, n4, n5, n6 denotam as três entidades do tipo nome_próprio que integram o conjunto n.

6. Conclusão e trabalhos futuros

Ao longo deste artigo foi descrito um modelo de dados que, utilizando o conceito de tipo de dados e construtores, permite uma modelagem da realidade mais simples. As decisões do analista de Sistema de Informações ficam facilitadas por não ter que tomar decisões subjetivas sobre a classificação dos objetos da realidade em diferentes classes parcialmente sobrepostas de conceitos no modelo.

O trabalho de pesquisa está se desenvolvendo em diversas frentes: especificação formal do modelo de dados, desenvolvimento de uma metodologia de projeto de esquema de banco de dados utilizando os conceitos do Modelo E e estudos para implementação de um SGBD.

Bibliografia

[Che 76] PPS Chen, "The entity relationship model: toward a unified view of data.", ACM TODS, v 1, n 1, pag 9, 1976.

[Cod 70] EF Codd, "A relational model of data for large shared data banks.", Communications of ACM, v 13, n 6, pag 377, 1970.

[Cod 79] EF Codd, "Extending the database relational model to capture more meaning.", ACM TODS, v 4, n 4, 1979.

[CODASYL 71] Data Base Task Group of CODASYL Programming Language Commitee. Report, apr 71.

[LPV 83] M. Lopez, J. Palazzo Oliveira & F. Velez, "the TIGRE Data Model.", relatório de pesquisa Tigre n 2, IMAG, Grenoble, França, nov 83.

[HM 81] M Hammer & D McLeod, "Database description with SMD: a semantic database model.", ACM TODS, v 6, n 3, pag 351, 1981.

[Mar 78] T. DeMarco, "Structured analysis and system specification.", New York, Yourdon Inc. , 1978.

[SNF 80] CS Santos, EJ Neuhold & AL Furtado, "A data-type approach to the entity-relationship model.", in Entity-relationship approach to systems analysis and design, ed PPS Chen, North Holland, 1980.

[SS 77] JM Smith & DCP Smith, "Database abstractions: aggregation and generalization.", ACM TODS, v 2, n 2, pag 105, 1977.

Apêndice

Definição de tipos de entidade

```
nome_próprio.

endereço.

salário.

idade.

equipamento.

tarefa.

funcionário = produto de (
    NOME: nome-próprio,
    RESIDENCIA: endereço,
    IDADE: idade,
    REMUNERACAO: salário).

engenheiro= especialização de (funcionário).
    FORMACAO: ramo-de-engenharia,
    DATA-FORMATURA: data).


técnico  = especialização de (funcionário).
   ESPECIALIDADE: especialidade-técnica).

máquina = especialização de (equipamento).

membro-de-equipe = soma de (engenheiro, técnico).

equipe = correspondência de (membro-de-equipe).

datilógrafo = especialização de (funcionário).

departamento = produto de (
   CHEFIA: funcionário,
   CORPO-TECNICO: equipe,
   INFRA-ESTRUTURA: correspondência de (equipamento)).

projeto = produto de (
   LIDER: engenheiro,
   PLANO: correspondência de (tarefa),
   EQUIPE: equipe,
   RECURSOS: correspondência de (máquina)).
```

REDES LOCALES DE COMPUTADORES PERSONALES: Una guía práctica para el profesional

Diego Passadore
Jorge Faral
Montevideo - *Uruguay*

1. INTRODUCCION

Las redes locales de computadores personales son una forma de proporcionar acceso compartido a recursos informáticos, de una manera conceptualmente óptima en el uso de las tendencias tecnológicas actuales. Existen diversas topologías, métodos de acceso, sistemas de transmisión y diseños de software, que están siendo desarrollados para lograr la completa funcionalidad que promete una red local, pero ninguno de ellos es la panacea. La pluralidad de soluciones disponibles es buena tanto para el profesional como para el usuario, porque la ausencia de una solución única, incentiva al profesional a encontrarla y da al usuario la posibilidad de comprar una red altamente especializada, que se ajuste a la medida de sus requerimientos.

Desafortunadamente, la diversidad significa que los estándares son pocos (o demasiados, depende de donde se los mire), lo cual es un problema particularmente fastidioso para la selección del software.

Este trabajo intenta ser una guía básica sobre algunos de los temas que debe conocer el profesional en computación acerca de las redes locales.

Se espera que la discusión de factores físicos y algunas consideraciones sobre factores lógicos podrán auxiliar a la selección de una red local y proteger de imprevistos una vez que se la ha instalado.

Los autores tienen el propósito de ampliar y completar el tema en futuros trabajos, y realizar una exposición de las características básicas de todas las redes locales de microcomputadores que se venden

en el medio. También desean dejar constacia que los conceptos vertidos no son fruto de vasta experiencia sino del estudio de la bibliografía utilizada (ver anexo), tal vez con la originalidad de sintetizarla en forma de conceptos comparados, clarificando al mismo tiempo la nueva terminología.

Por razones prácticas se han dejado de lado temas de interés e importancia como pueden ser: el uso de PBX ("Private Branch eXchange") como red local de transmisión de datos; planificación, instalación y administración de redes; estándares IEEE.

2. TECNOLOGIAS COMPARADAS

2.1 TOPOLOGIAS

Se denomina topología de una red, a la configuración o forma que adquieren los nodos y las diferentes interconexiones que los unen. Existen tres maneras básicas de interconectar computadores personales en red local: en estrella ("star"), en anillo ("ring") y en bus.

Estrella: es la topología de red local en la cual todas las estaciones están conectadas a una estación central que establece, mantiene e interrumpe las comunicaciones en al red.

Anillo: cada estación es conectada a otras dos estaciones, repitiéndose este proceso hasta cerrar un circuito. Los datos son transmitidos de un computador a otro y siempre en la misma dirección. Esta topología ha evolucionado con la incorporación de centros de cableado ("wire centers") que la hacen parecida a un conjunto de estrellas interconectadas. De esta forma incorpora todas las ventajas topológicas de las estrellas (de aquí en más se entenderá por anillos a los anillos con centro de cableado).

Bus: todas las estaciones se vinculan a un único medio de transmisión de manera que todas escuchan todas las transmisiones en la red.

Cada topología se acomodará mejor a determinados medios de transmisión, tendrá estrategias óptimas de ruteo o de métodos de acceso y se identificará con características de confiabilidad propias.

2.1.1 RUTEO Y METODO DE ACCESO

Se denomina ruteo al proceso de determinar el reenvío de un mensaje hacia su destino. Método de acceso es la técnica para determinar y controlar cuál de las estaciones que desean transmitir será la siguiente en hacerlo.

Estrella: permite un ruteo excepcionalmente fácil porque el nodo central conoce el camino a los otros nodos. El acceso puede ser fácilmente controlado y pueden asignarse prioridades de transmisión. A su vez estas ventajas funcionales exijen que la unidad central posea la capacidad computacional adecuada.

Anillo: no requiere ruteo ni control ya que cada nodo siempre pasa el mensaje en la misma dirección. Esta forma de transmisión tiene la característica singular de poder proporcionar "verificación de recepción". Esto es posible agregando un bit al mensaje, que es complementado en la recepción. Como al final el anillo retorna al que lo envió, el estado de dicho bit puede ser verificado para confirmar su recepción. También puede ser examinada la "redundancia cíclica" del mensaje, para confirmar que no hubo error de transmisión. Esta topología, utiliza casi siempre el método de acceso token-passing (ver apartado 2.2).

Bus: no se realiza ruteo porque es un medio "broadcast" en el cual todos los nodos reciben todas las transmisiones, y todos compiten entre sí por el uso del medio. Este esquema distribuye el control entre todos los nodos. Requiere un medio de transmisión full-duplex. En la mayoría de los casos utiliza el método de acceso CSMA/CD y en algunos pocos el token-passing (ver apartado 2.2).

2.1.2 MANTENIMIENTO, INSTALACION Y PRUEBA

Estrella: El nodo central permite una gran eficiencia en el mantenimiento y prueba de la red.

Anillo: el centro de cableado simplifica estas tareas a través de relevadores ("bypass") electrónicos. Estos permiten que el tráfico de mensajes viaje entre dos nodos que no están físicamente adyacentes, mientras una estación es instalada o queda fuera de servicio. Por otra parte este esquema aumenta la longitud del cableado.

Bus: la adición y caída de nodos puede realizarse sin interrumpir el tránsito en la red. Es la topología que menos cantidad de cable utiliza.

2.1.3 CONFIABILIDAD

Entendemos por confiabilidad la seguridad con que se desempeña un sistema una vez que alguien está usándolo.

Estrella: al haber control centralizado se requiere que el nodo central sea excepcionalmente confiable.

Anillo: es muy buena debido a los relevadores electrónicos del centro de cableado que permiten derivación automática.

Bus: la ausencia de ruteo y de control centralizado proporciona alta confiabilidad.

2.2 METODOS DE ACCESO: CSMA/CS vs. TOKEN PASSING

Hay dos formas de control de acceso a una red:
- los sistemas de circuitos conmutados adquieren el acceso al medio de transmisión y envían información de direccionamiento solamente al comienzo de un llamado, y varios mensajes pueden ser intercambiados durante el llamado.
- los sistemas de paquetes conmutados requieren que los mensajes sean partidos en paquetes y que el medio sea accedido para cada paquete enviado.

Para la transmisión de datos, la conmutación de paquetes es mucho más eficiente que la conmutación de circuitos, porque un circuito puede ser usado para mensajes entre varias parejas de comunicadores simultánemente.

Dentro de los sistemas de paquetes conmutados los métodos de control más populares son el token passing y el CSMA/CD.

Token passing: es una variación eficiente del polling. Un patrón especial de bits llamado token circula a través de la red. Si un nodo no tiene nada para transmitir, permite pasar el token. Si el nodo tiene algo para transmitir toma el token e inserta un mensaje delante de token.

CSMA/CD: utiliza la técnica de contención, específicamente acceso múltiple de detección de portadora, o CSMA ("Carrier Sense Multiple Access"). En CSMA, un nodo escucha al medio antes de transmitir; si nada es escuchado comienza a transmitir. Sin embargo hay una probabilidad finita que otro nodo tome la misma decisión al mismo tiempo y dos o más transmisiones comiencen simultáneamente. Por esto, un

dispositivo adicional, de detección de colisión es agregado a los sistemas CSMA, creando el CSMA/CD. La detección de colisión se logra comparando los datos transmitidos con los recibidos, y viendo si el mensaje del medio coincide con el que se transmitió. En tal caso se utiliza la vuelta atrás. La cantidad de tiempo antes de reintentar puede ser aleatoria o seguir la regla "vuelta atrás exponencial" (se va duplicando la media del número aleatorio). Teóricamente no hay garantía de que un nodo tenga la posibilidad de transmitir en algún momento, aunque este problema puede ser ignorado en sistemas con menos del 40 % de su capacidad.

2.2.1 CONTROLADOR DE INTERFACE CON LA RED

El controlador de interface con la red es un circuiterío electrónico que conecta la estación con la red. El circuiterío determina cuándo una estación debe transmitir, detecta el arribo de mensajes, indica condiciones de error, y puede incluir memoria transitoria para guardar mensajes transmitidos y recibidos. A continuación será descrita la lógica que utiliza el controlador de interface con la red en cada método de acceso.

Token passing: está basado en el principio de que el permiso para usar el medio de comunicación está dado de forma de un token que pasa de estación en estación: la lógica de transmisión debe esperar el aviso de la lógica de recepción, de que el token ha sido recibido; la transmisión comienza luego que el token ha sido apropiadamente tratado. Suponiendo que el mismo se puede detectar con un indicador de un bit, la demora introducida por una estación es aproximadamente de un bit, o sea el tiempo para examinar, copiar o cambiar un bit según sea lo necesario. Luego de encontrado el token, la estación lo altera para formar otro patrón especial llamado colector, agrega el mensaje que desea mandar, y luego incluye el token. Este token sólo se agrega luego de ser recibido el colector por el nodo que lo envió. Esto asegura que un solo token o colector está en el anillo en un momento dado, simplificando la recuperación de errores y mantieniendo un rendimiento prácticamente óptimo. En el tiempo, apenas una estación comenzó a transmitir, ya está recibiendo su propia transmisión con demora de sólo unos pocos bits.

CSMA/CD: la lógica de transmisión debe esperar la condición "no portadora", y la expiración del "time out" si una colisión ocurrió en un intento de transmisión previo. Después del comienzo de la transmisión hay que controlar la detección de colisión, y cuando ésta es detectada se envía

un patrón de atascamiento para asegurar la detección en toda la red. Después de cierto tiempo el que transmite debe tratar de acceder el medio nuevamente. La recepción comienza cuando se detecta un patrón especial que condiciona el circuiterío para una adecuada recepción de los subsiguientes mensajes de datos. Este patrón inicial es conocido como preámbulo. El circuiterío de recepción reconoce el fin de un preámbulo y trata los apropiados bits que siguen como dirección. Nótese que la tarea descripta anteriormente es realizada por todas las estaciones de la red, ya que hasta no obtener la dirección no saben si el mensaje está dirigido a ellas o no. Las características apuntadas, exigen que exista una alta velocidad de propagación de las señales a través del medio de transmisión (ver apartado 2.4).

2.2.2 CONFIABILIDAD

Token passing: provee la seguridad y la insensibilidad a la distancia del sistema de polling, pero puede utilizar el medio más eficientemente ya que el token es pasado a través en vez de venir siempre de una estación maestra. Sin embargo la propagación del token descansa en el correcto funcionamiento de los nodos y se deben tomar previsiones para recuperar exitosamente fallas que causen la desaparición del token.

CSMA/CD: tiene la ventaja que el control está distribuído en los nodos, y es posible lograr un alto grado de confiabilidad. CSMA/CD requiere que los mensajes no sean menores a un largo mínimo, que está en función de la velocidad de transmisión y el largo del medio.

2.2.3 MANTENIMIENTO DE LA RED: CIRCUITOS ABIERTOS

Un circuito abierto es un arreglo de componentes eléctricos y cableado a través del cual no puede fluir la corriente porque el cableado está desconectado en algún punto, o porque un componente eléctrico ha fallado de manera que no puede fluir la corriente.

Token passing: aunque un circuito abierto dentro de un centro de cableado es muy raro, un relevador de derivación ("bypass relay") defectuoso podría causar dicha falla. Más probable es que el cableado a una estación se haya roto. Esa rotura pone el relevador de derivación en modo derivación y la operación de la red continúa. La técnica más usada de relevador de derivación es un anillo alternativo entre los centro de cableado o un cambio de sentido en el anillo existente, lo que permite a éstos seguir serialmente consecutivos y a las estaciones seguir lógicamente consecutivas.

CSMA/CD: un circuito abierto en un transmisor-receptor podría causar la falta de datos recibidos (y falta de detección de la portadora). Un circuito abierto en un cable coaxial podría crear una discontinuidad en la impedancia que podría causar reflexiones de señales que hacen que el número de colisiones crezca rápidamente.

2.2.4 MANTENIMIENTO DE LA RED: CIRCUITOS CORTOS

Un sistema eléctrico en el cual la corriente fluye directamente de un conductor al siguiente sin pasar a través del dispositivo(s) que supuestamente reciben la corriente, es un circuito corto.

Token passing: es muy difícil que ocurra un circuito corto dentro de un centro de cableado. Más probable es que el cableado a una estación haya sido cortado.

CSMA/CD: un circuito corto en un transmisor-receptor o un cable coaxial podría causar fallas similares a las de circuitos abiertos.

2.2.5 MANTENIMIENTO DE LA RED: MAL FUNCIONAMIENTO DEL CONTROLADOR

Token passing: la falla más común es la desaparición del token. Para detectar esta ocurrencia algunos sistemas confían en una "estación central" que revise la aparción periódica del token; otros confían en cada estación para monitorear el medio por banderas de mensajes y tokens. En ambos casos una vez que una pérdida de token es detectada, uno nuevo es generado.

CSMA/CD: si el circuiterío que detecta la portadora falla en ON el controlador nunca transmitirá, creyendo que alguien más está transmitiendo. Si falla en OFF la estación transmitirá cada vez que hay algo para mandar, quizás creando una colisión. Otra falla posible ocurre en el sistema de detección de colisiones. Un dispositivo útil es el detector de colisiones tardías: una colisión que tarde en ocurrir más tiempo que el de una vuelta completa indica que alguna estación no está efectuando la revisión para detectar portadoras apropiadamente, o la red tiene una demora de vuelta completa más larga que lo que debería.

2.2.6 SOBRECARGA DE LA RED

Token passing: el tiempo es desperdiciado esperando que el token atraviese la red y arrive a la estación que desea transmitir. En promedio, el token estará del lado opuesto del anillo de la estación que quiere transmitir. Es el método más eficiente cuando la red está cargada y el token tiene que ir de estación activa en estación activa.

CSMA/CD: el tiempo es desperdiciado cuando ocurre una colisión. Es el método más eficiente cuando no hay colisiones, ya que está pronto instantáneamente para transmitir. Las demoras se vuelven severas con un 50% de la carga.

2.2.7 RECONFIGURACION

Token passing: los anillos token pueden ser instalados con conexiones no usadas, a través de relevadores de derivación. Una estación adicional, u otro centro de cableado puede ser agregado y probado antes de la interrupción momentánea causada por la abertura del relevador de derivación.

CSMA/CD: los sistemas que utilizan derivadores de penetración ("derivation taps") pueden tener estaciones agregadas con sólo una interrupción momentánea, al ser conectado con el cable coaxial el transmisor-receptor asociado a la nueva estación. La estación puede hacer una revisión por sí misma antes de ser agregada al sistema.

2.2.8 EXTENSION DE LA RED

Token passing: en redes más grandes se experimenta demoras más grandes, especialmente si es el número de estaciones el que crece, ya que cada estación contiene memorias intermedias donde es analizado el contenido de los mensajes del medio.

CSMA/CD: cuando la distancia de la red crece, necesita un mayor tamaño de paquete mínimo, de manera que todas las colisiones puedan ser detectadas. Una mayor longitud de paquete mínimo decrece el rendimiento efectivo.

2.3 SISTEMAS DE TRANSMISION

Se entiende por sistema de transmisión la manera o método con el cual las señales son adaptadas al medio de transmisión (ver apartado 2.4).

Actualmente se utilizan dos tipos de sistemas: baseband (banda simple o base) y broadband (banda ancha).

Baseband: las señales se adaptan al medio de transmisión sin ser cambiadas de frecuencia. En un sistema baseband hay siempre un solo conjunto de señales en un solo rango de frecuencias (por ejemplo: la señales de voz en el rango de 300 a 3000 Hz, aparecerán en el medio de transmisión en el rango de 300 a 3000 Hz).

Broadband: la señales se adaptan al medio después de ser cambiada su frecuencia. En un sistema broadband, hay varios conjuntos diferentes de señales, presentes a un tiempo en el medio de transmisión, trasladados cada uno a rangos de frecuencia sin interferecia (por ejemplo las señales de voz humana podrían aparecer en el rango de 2.000.300 a 2.003.000 Hz). Por esta razón este sistema se utiliza para transmitir diferentes tipos de información: datos, voz e imagen.

Indudablemente las prestaciones del sistema broadband son comparativamente muy superiores a las del baseband pero a un costo mayor.

2.4 MEDIOS DE TRANSMISION

Un medio de transmisión es el medio físico usado para propagar una señal eléctrica que representa información.

Todas las topologías y métodos de acceso vistos hasta el momento son combinables entre sí sobre una gran variedad de medios de transmisión.

Discutiremos las propiedades físicas de las tres categorías de medios que se adaptan mejor a las redes locales: par trenzado ("twisted pair"), coaxial y fibra óptica. Su costo está en relación directa a las prestaciones.

Par trenzado: son dos cables aislados enrollados uno en el otro uniformemente de manera que cada uno está igualmente expuesto a las interferencias eléctricas del ambiente. El par puede estar a su vez blindado o recubierto por otro aislante.

Coaxial: es un medio de transmisión eléctrica en el que un cable está envuelto por aislante y éste a su vez por un conductor tubular de metal cuyos ejes de curvatura coinciden con el centro del cable.

Fibra óptica: medio de transmisión consistente en fibras de vidrio muy delgadas. En un extremo de la fibra un diodo emisor introduce luz y ésta va reflejándose en el interior de la superficie de la fibra hasta que llega a el otro extremo, donde un detector convierte la luz en una señal eléctrica.

2.4.1 VELOCIDAD DE TRANSMISION

Par trenzado: hasta 10 Mbs (mega bits / segundo). A muy alta velocidad conviene que sea blindado.

Coaxial: hasta 100 Mbs.

Fibra óptica: hasta 1.000 Mbs.

2.4.2 ADAPTABILIDAD A LOS SISTEMAS DE TRANSMISION

Par trenzado: se adapta mejor al sistema baseband.

Coaxial: se adapta a baseband y broadband. En sistemas broadband es la tecnología usada por la TV a cable.

Fibra óptica: especialmente utilizada en sistemas broadband.

2.4.3 CARACTERISTICAS ELECTRICAS

Par trenzado: tiene excelente inmunidad a las interferencias si es blindado.

Coaxial: inmunidad excelente a las interferencias; permite un acho de banda mayor que el par trenzado.

Fibra óptica: interferencia eléctrica prácticamente nula; ancho de banda muy grande; niveles muy bajos de señal que requieren el uso de convertidores lógicos.

2.4.4 INSTALACION

Par trenzado: es fácil de cortar, desmontar y unir. Es el medio común de transmisión telefónica.

Coaxial: más difícil y costosa. Tiene sección mayor que el par trenzado y más rigidez. En baseband, debe cuidarse una adecuada separación entre las estaciones para evitar la reflexión de señales, debidas a irregularidades de impedancia. En broadband, para extensiones grandes, la instalación debe ser precedida por una cuidadosa

planificación: deben calcularse los niveles de señal en toda la red, lo cual requiere conocer con precisión el largo de los cables y la ubicación de las derivaciones actuales y futuras y por tanto la ubicación de los amplificadores de señal.

Fibra óptica: muy fácil de instalar por su pequeña sección, ductilidad y fidelidad de transmisión.

2.4.5 ADAPTABILIDAD A LAS TOPOLOGIAS Y METODOS DE ACCESO

Par trenzado: anillos y bus.

Coaxial: bus.

Fibra óptica: por su bajo nivel de señal se hace muy difícil la detección de colisiones en un bus con CSMA/CD. Puede ser usada eficientemente para la conexión entre segmentos de bus y entre centros de cableado.

3. SOFTWARE

En este tema se destacan dos aspectos principales: software de la red y software adquirido que utiliza la red.

3.1 SOFTWARE DE LA RED

Hay que investigar de cerca los dispositivos y el rendimiento del software de la red local antes de pensar en otros factores como el precio o la facilidad de instalación. Hay que pensar en el software de redes locales como otro ambiente de programación con el cual se debe trabajar: como un sistema operativo, BASIC o planillas electrónicas.

El software de red local agregará sus propias restricciones a aquellas del sistema operativo y dará más posibilidades de las que se obtiene con un sistema monousuario. Se debe interactuar con el software de red local de la misma manera que con el sistema operativo. Y esta interacción puede ser muy pesada o totalmente transparente.

3.1.1 MODOS DE COMPARTIR RECURSOS

Datos: hay que estudiar detenidamente la descripción para compartir dispositivos de disco, porque se brindan muchas posibilidades diferentes. La distinción entre

compartir discos y compartir datos o archivos es muy importante (ver apartado 3.1.2).

Impresoras: si las redes son para compartir, compartir las impresoras debería tener alta prioriadad por su alto costo y poco frecuente uso. El modo más simple de usuarla es el dedicado, en cuyo caso todos los demás usuarios que quieran utilizarla van a obtener "recurso ocupado". En los software más sofisticados el "spooling" de impresora es parte integral del compartir de impresoras.

Comunicaciones: se logran extensiones de la red a través de puentes ("bridges") y compuertas ("gateways"). Un puente permite intercambiar mensajes entre dos redes locales con igual protocolo. Una compuerta permite intercambiar mensajes entre una red local y otra red con un protocolo totalmente diferente, haciéndose la conversión de protocolo en la misma compuerta.

No son detalladas otras funciones de servicio menos básicas como son: correo electrónico, contabilidad, respaldo, etc.

3.1.2 RELACION SISTEMA OPERATIVO - SOFTWARE/HARDWARE DE RED Y SERVIDORES

La interacción con el sistema de computación está controlada por el sistema operativo. Este interpreta los comandos, maneja la memoria de disco y en general supervisa el sistema entero. Pero el control del sistema operativo es limitado: convencionalmente está diseñado para controlar un solo computador y nunca va más allá de los confines del circuiterío del computador; los discos están bajo su control e inaccesibles para otros computadores; es más, los discos de otros computadores están inaccesibles para su computador. Pero cuando un computador es parte de un red mucho de esto cambia. El software de control de la red agrega rutinas extras, permitiendo interactuar con el resto de la red. Al igual que el sistema operativo el módulo de la red opera en "background" monitoreando constantemente lo que pasa dentro del computador. Revisa para ver si una actividad particular de su programa de aplicación requiere ir más alla del computador y comunicarse con el resto de la red. Cuando se hace un pedido local es pasado directamente a la máquina local. Pero cuando se hace un pedido que requiere actividad de red, el software de red envía el comando a la máquina designada para procesarlo.

La mayoría de las redes tienen uno o más computadores especiales que manejan la mayoría de los comandos y

funciones de la red. A estos computadores se les llama servidores de la red ("servers"), los cuales permiten compartir los recursos disponibles (ver apartado 3.1.1). La mayor diferencia entre los tipos de servidor es si el software de red procesa los comandos antes o después de que los maneje el sistema operativo.

Merecen especial atención los servidores de información o de datos que describimos a continuación:

Servidor de disco: es algo más que un disco duro que pueda ser accedido por más de un computador encadenado a la red. Los pedidos de red son manejados por cada estación exactamente igual que si fueran hechos a un disco en un computador monousuario. El disco duro compartido aparace a la estación individual del sistema como nada más que otra tarjeta controladora de disco, como si fuera otro dispositivo de memoria donde los archivos pueden ser leídos o grabados. En la implementación de servidor de disco el sistema operativo mantiene el control. Algunos sistemas operativos mantienen una copia del directorio del disco o FAT ("File Allocation Table") en memoria. Esto crea un problema cuando varios computadores trabajan en el mismo archivo, ya que cada estación actualiza su copia del FAT independientemente en su propia memoria, no teniendo en cuenta los cambios hechos por otros en la red. Un método común para minimizar este problema es dividir el disco del servidor en varias particiones o volúmenes de usuarios, cada uno de los cuales actúa como un disco lógico independiente para la red. Estas particiones pueden ser de largo fijo o variable usualmente con algunas restricciones de tamaño. Los volúmenes (particiones) son dedicados a las estaciones de la red en forma exclusiva (un volumen pertenece a no más de una estación). Pero pueden haber volúmenes públicos que sean sólo de lectura.

Servidor de archivo: en vez de aparecer la red a cada estación como discos adicionales, las estaciones acceden a la red a través de archivos individuales. En vez de operar bajo el control del sistema operativo, el software adicional actúa como filtro de todos los pedidos hechos por los programas de aplicación. Los pedidos de recursos externos son enviados directamente a la red por el filtro de la red, y nunca llegan al sistema operativo de la máquina. La red rutea los comandos al servidor de archivo y ellos son completados por el sistema operativo del servidor. Aquí dos o más usuarios pueden compartir un mismo archivo. Es responsabilidad del DBMS trancar otros usuarios cuando un usuario esté cambiando un campo o un registro, permitiendo de esta forma mantener la integridad de la base de datos.

3.2 SOFTWARE ADQUIRIDO QUE UTILIZA LA RED

Ante un software DBMS, planilla electrónica o correo electrónico que dice soportar redes locales, se podría pensar que va a funcionar en la red como a uno le gustaría. En realidad, es muy probable que se lleve una gran desilusión. Si uno interpreta, por ejemplo, que el DBMS va a proveer tranca a nivel de registro, quizás suceda que el software se comporte adecuadamente en la red sólo como múltiples sistemas monousuario, es decir, que exista tranca sólo a nivel de archivo.

Otro problema importante es el de copias protegidas y licencias del software. La mayoría de los paquetes de software no están diseñados para redes y legalmente se requiere una copia separada para cada estación de trabajo.

A través de los paquetes de software puede quedar en evidencia la verdadera potencia de una red, pero dado que la compatibilidad entre redes y paquetes de software de terceros se podría considerar aún tenue, lo mejor sería no pagar por nada (ni red local, ni paquetes de software), hasta que todo trabaje satisfactoriamente.

4. RECOMENDACIONES

Como a menudo sucede: "lo mejor es enemigo de lo bueno", y las mejores soluciones en redes locales se ubican entre los extremos de lo más barato y lo más eficiente. Quizás se eviten muchos errores, si se tiene en cuenta en todo momento, que la red debe servir al usuario, y no al revés. Por ejemplo, la topología de cualquier red que se compre, debería ser invisible al usuario: debe interesar la función y no la forma. Para esto hará falta no sólo conocer sobre redes sino tanto o más sobre el usuario: cómo es; cómo está organizado; qué y cuánta información necesita de quién y para quién; cómo procesa la información y qué software utilizaría; cuánto va a crecer.

Tal vez la recomendación sea que ...
no necesita una red local, sino un sistema convencional de tiempo compartido, o quizás una solución híbrida.

Podría incluirse una extensa lista de recomendaciones vinculada a temas que expresamente se han dejado fuera del alcance de este trabajo. Sin embargo, se ha querido indicar lo apuntado anteriormente por tratarse de un criterio fundamental de prudencia a tener en cuenta para instalar cualquier sistema de información.

5. CONCLUSION

Se está, al parecer, en la primera generación de las redes locales caracterizadas por una gran rigidez debida a la dependencia de sus componentes hardware y los estándares aún en evolución.

Desde el punto de vista estrictamente técnológico, obviando funcionalidades del software y aspectos comerciales, se distinguen dos estándares en pugna: bus/coaxial/CSMA-CD vs anillo/par-trenzado/token-passing. comercial,

GLOSARIO

Contención Método que, teniendo múltiples estaciones accediendo al medio de transmisión, hace que cada estación decida, en una base de igualdad con otras estaciones, cuándo transmitirá.

Derivador de penetración Dispositivo usado para conectar un transmisor-receptor al bus sin requerir que el bus sea interrumpido para los ajustees de instalación. Esto se logra por un dispositivo en forma de aguja que penetra en el aislante del bus (un cable coaxial) y alcanza al centro del conductor del coaxial.

PBX Paquete de conmutación telefónica privada que sirve a una compañía, y se conecta a la red nacional de teléfonos.

Relevador de derivación Relevador de una red anillo que permite que el tránsito de mensajes viaje entre dos nodos que no están normalmente adyacentes. Usualmente esos relevadores están ordenados de manera que cualquier nodo pueda ser removido de la red para mantenimiento, y los dos nodos a ambos lados del nodo removido estén ahora conectados vía el relevador de derivación.

Transmisor-receptor Dispositivo que usa datos digitales para crear una señal que puede ser transmitida a través de una gran distancia, y puede recibir dichas señales determinando qué datos digitales fueron usados para crearlos. Difieren de los modems en que no necesitan modular la portadora.

BIBLIOGRAFIA

BARTIMO, JIM — A network perspective, Focus CW, Julio 1984.

BIDAL, GERARD — La seconde vague d'Ethernet, Le Monde Informatique, 31 de Marzo de 1986.

Reseaux locaux: premieres certitudes, Le Monde Informatique, 7 de abril de 1986.

BIRENBAUM, LARRY — The IBM PC meets Ethernet, BYTE, Noviembre 1983.

COOPER, EDWARD — Local-area network installation: Plan now or pay later, Focus CW, 1985.

DERFLER, FRANK — The lay of the LANs, PC Magazine, 5 de febrero de 1985.

DOUNIS & EFROYMSON — War stories from the network front, Computerworld, 25 de setiembre de 1985.

EDP ANALYZER — Keeping abreast of telecommunications, marzo 1986.

ELGAR, GEORGE — Gateways, Computerworld, 25 de setiembre de 1985.

GOETZE, EARL — IBM's Token-passing LAN protocol supports major SNA functions, Computer Technology Review, winter 1983.

GOLDHABER & ROSCH — Networks at your service, PC Magazine, 5 de febrero de 1985.

IBM — APPC/PC Programming Guide SC 30-3396.

LE MONDE INFORMATIQUE — Systeme multi-utilisateur contre reseau local, 12 de mayo de 1986.

LIDDLE, DAVID — After the storm, Focus CW, julio 1984.

LYTEL, DAVID — Gearing software to networks, Computerworld, 25 de setiembre de 1985.

MARINHO, JOSE — Redes locais de computadores, Cuadernos de tecnologia, Digibrás Nro. 6.

MC NAMARA, JOHN Local area networks:an introduction to the technology, Digital Press, enero 1985.

METCALFE, ROBERT Local networking of personal computers, IFIP 1983.

MIER, EDWIN The evolution of a standard Ethernet, BYTE, diciembre 1984.

PEREIRA, LEONARDO Redes locais de computadores, SUCESU, octubre 1981.

POTTER, DAVID The token ring, Focus CW, julio 1984.

PRINCE, VIOLAINE Quelle strategie pour les directions informatiques?, Le Monde Informatique, 14 de abril de 1986.

Passerelles, Le Monde Informatique, 5 de mayo de 1986.

SEMILOF, MARGIE The ubiquitous twisted pair, Focus CW, marzo 1985.

SCOTT, J. Local-area networks for the IBM PC, BYTE, diciembre 1984.

STROLE, NORMAN A local communications network based on interconnected token-access rings: a tutorial, IBM J. Res. Develop., setiembre 1983.

TERRIE, DAVID Local-area networks, Focus CW, marzo 1985.

TEXAS INSTRUMENTS Local Area Network Products, Seminar Hanbook, 1985.

VAZQUEZ, JOSE Automatización de oficinas, Cuadernos de informática, Eeria nro. 4 1985.

WECKER, STUART Two games in town, Focus CW, agosto 1985.

Combining voice and data through local nets and PBX, Focus CW, setiembre 1985.

YARMIS, JONATHAN Networking software, Focus CW, marzo 1985.

ZAK, MICHAEL Charting the waters, Focus CW, julio 1984.

DISEÑO DE UN LENGUAJE PARA BASES DE DATOS FUNCIONALES

José de Jesús Pérez Alcázar
Alberto Henrique Frade Laender
Roberto da Silva Bigonha
Departamento de Ciência da Computação
Minas Gerais - *Brasil*

RESUMEN

Este articulo describe las principales caracteristicas de un lenguaje llamado LBF (Lenguaje para Bases de Datos Funcionales) el cual es un lenguaje auto-contenido para el manejo de bases de datos funcionales. LBF es una extensión del lenguaje DAPLEX definido por Shipman. Entre las extensiones podemos enumerar la inclusión de comandos de entrada y salida, comandos condicionales, operandos aritméticos y lógicos y comandos para modificar el esquema.

1. INTRODUCCION.

En los últimos años, varios modelos de datos han sido propuestos con el propósito de capturar mas semántica en los datos y remover varias de las restricciones impuestas por los llamados modelos clásicos (jerárquico, de red y relacional). Estos modelos, denominados modelos semanticos [4,15], fueron diseñados para ofrecer mecanismos de modelaje mas ricos y expresivos, permitiendo la descripción de la estructura de una base de datos de una forma más clara y precisa [3,7,8,18,21].

Entre los modelos de datos semánticos descritos en la literatura se destaca, por su simplicidad, el modelo funcional [6,9,12,19,27]. De las propuestas sobre el modelo funcional existentes, algunas incorporan lenguajes de manejo de datos a través del uso de funciones y conjuntos de operadores. e estas, solamente las propuestas de Shipman [19] y Buneman & Frankel [6] integran operaciones de manejo de datos con operaciones de propósito general en un solo lenguaje. Esta es una caracteristica importante ya que en general los modelos de datos semánticos propuestos en la literatura facilitan solamente el modelaje de las propiedades estáticas de las aplicaciones (estructuras), prestando poca atención a los aspectos dinámicos (operaciones) [15].

Seguramente, el modelo de Shipman [19] es el más destacado de los modelos funcionales. Este modelo utiliza varios de los recientes resultados provenientes de estudios en el area de modelaje de datos, lo cual convierte a DAPLEX, su lenguaje de definición y manipulación de datos, en un lenguaje bastante rico semanticamente.

El propósito de este articulo es describir un lenguaje del tipo de DAPLEX que está siendo desarrollado en el Departamento de Ciencias de la Computación de la Universidad Federal de Minas Gerais. Este lenguaje, llamado LBF (Lenguaje para Bases de Datos Funcionales), es una extensión de DAPLEX, en el cual fueron incluidas algunas construcciones propuestas por KulKarni y utilizadas en EFDM [13], además de facilidades para operaciones de E/S y comandos condicionales.

A seguir el artículo se divide en cuatro secciones. En la sección 2 describimos el modelo de datos funcional y los motivos de su selección para el desarrollo de este trabajo. En esta sección también presentamos una descripción de la propuesta de Shipman. En la sección 3 describimos LBF, sus estructuras y operaciones y damos un pequeño ejemplo de su utilización. En la sección 4 describimos el estado actual del trabajo y mostramos algunas de las orientaciones para estudios futuros. En la sección 5 presentamos algunas conclusiones. Por último anexamos un apéndice con la sintaxis de LBF.

2. EL MODELO DE DATOS FUNCIONAL.

El modelo funcional nace como resultado de la busqueda de nuevas formas de modelaje que sirvan de base común o formalismo en la descripción y comparación de los tres modelos clásicos [20]. Este formalismo debería ofrecer una base unuficada para el diseño de esquemas tanto relacionales como de red, sin anomalías de actualización.

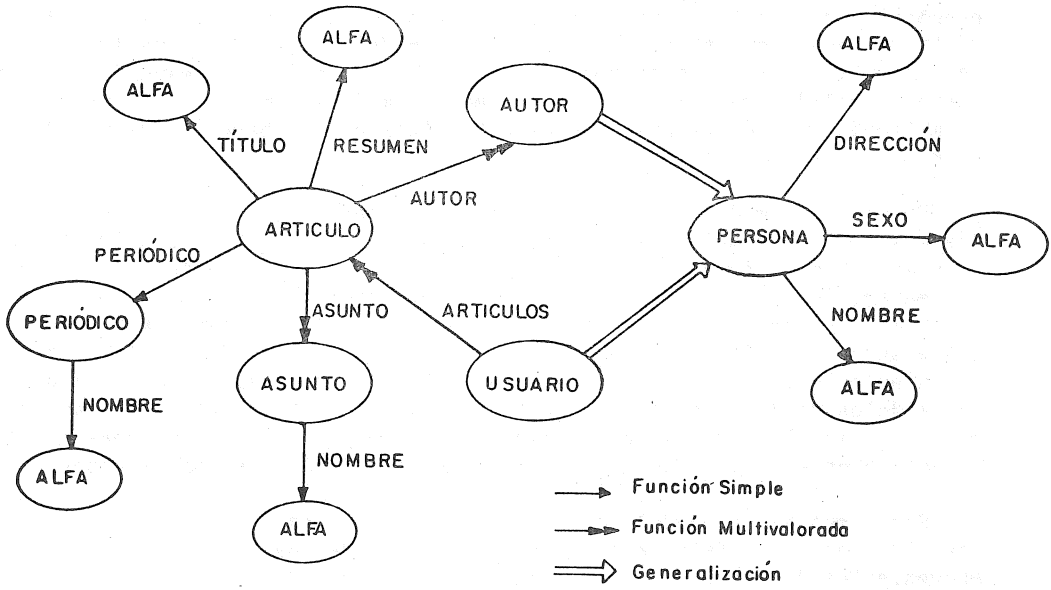


Fig. 1

El modelo funcional tiene sus bases en el concepto matemático de función y utiliza dos tipos de primitivas (estructurales) para el modelaje del mundo real [13]: los conjuntos de objetos y las funciones que son aplicadas a estos para relacionarlos entre si. Los conjuntos de objetos estan divididos en conjuntos de entidades y conjuntos de valores. La principal diferencia entre un conjunto de valores y un conjunto de entidades es que el primero

nunca es parte del dominio de una función. Los conjuntos de valores son llamados por Kulkarni [13] conjuntos de entidades predefinidas (ej: conjuntos de enteros, reales, etc). Las funciones pueden ser totales o parciales, y la distinción corresponde a una restricción [11] sobre el número de elementos del conjunto (dominio) que deben participar del relacionamento funcional.

La figura 1 muestra la descripción gráfica, de acuerdo con el modelo funcional, del esquema conceptual de una base de datos para una biblioteca personal de artículos publicados en revistas (periódicos). Esta base de datos será usada en la mayoría de nuestros ejemplos.

2.1. El Modelo de Datos Funcional de Shipman

Como fue dicho anteriormente, el modelo funcional mas destacado es el modelo propuesto por Shipman [19], el cual esta incluido en el lenguaje DAPLEX. Su objetivo primordial es ofrecer un lenguaje conceptualmente natural que sirva de interfaz en la utilización de una base de datos. Las construcciones básicas del lenguaje DAPLEX son las entidades que son usadas para modelar los objetos de la aplicación y las funciones que son usadas para modelar sus propiedades. El aspecto mas interesante del lenguaje DAPLEX es su simplicidad aliada a construcciones de modelaje semanticamente poderosas.

Entre las características principales del lenguaje DAPLEX podemos enumerar las siguientes:

1) Las funciones pueden tener cero o más argumentos. Las funciones sin argumentos son utilizadas para modelar entidades o objetos del mundo real y las funciones con argumentos son utilizadas para modelar atributos de entidades y relacionamentos entre estas.

2) Las funciones pueden ser clasificadas en: simples (retornan como resultado una entidad) y multivaloradas (retornan un conjunto de entidades).

3) Existe una diferencia entre entidad y su identificación externa o llave (orientación a objetos).

4) Las entidades puedem ser organizadas en una jerarquía de tipos [21,25], donde atributos y relacionamentos de los supertipos son automáticamente heredados por cada uno de los subtipos correspondientes.

5) A partir de funciones básicas es posible crear funciones derivadas, es decir, el concepto de información derivada [25] es soportado de forma simple y natural.

2.2. Una Validación del Modelo Funcional

Pero porqué el modelo funcional es una herramienta atrayente para el modelaje conceptual ?. Kerschberg & Pacheco [12], Kulkarni [13] y Orman [16] enumeran algunas de estas razones:

1) Los modelos funcionales ofrecen un ambiente de modelaje semanticamente rico. El hecho de no diferenciar entre datos y programas (información derivada) facilita bastante el modelaje conceptual.

2) Los lenguajes de consulta pueden ser especificados de una forma más natural. Además, facilidades de manejo de datos pueden ser integradas naturalmente con operaciones de propósito general (FQL, el lenguaje definido por Buneman & Frankel [6], es un ejemplo).

3) Existen algoritmos para mapear la estructura de una base de datos de red (CODASYL) o relacional para una funcional [12]. Como consecuencia, el modelo de datos funcional puede ser usado como mecanismo de integración de bases de datos hetereogeneas [22].

4) El modelo funcional representa la información a través de hechos atómicos (funciones) [16], eliminando algunos problemas de redundancia y evitando anomalias en la actualización.

5) La teoría matemática de funciones puede ser explorada para ofrecer una base teórica sólida para bases de datos.

6) La función segun Orman [16], es una estructura más primitiva y poderosa que otras comunmente usadas por los modelos ya conocidos, como relaciones, archivos, conjuntos DBTG y arreglos. A partir de funciones, consultas y restricciones son naturalmente implementadas, ya que consultas son funciones aplicadas a valores de entrada y que retornan valores de salida, al mismo tiempo que restricciones son predicados que corresponden a funciones cuyo codominio es el conjunto de valores lógicos.

3. DESCRIPCION DEL LENGUAJE LBF

LBF es un lenguaje interactivo, con el cual el usuario puede crear, operar y actualizar su base de datos. Este lenguaje, como el lenguaje EFDM [13], es auto-contenido, es decir no necesita ser enbutido en un lenguaje de propósito general como sugiere Shipman [19] al definir DAPLEX. LBF es básicamente una extensión del lenguaje DAPLEX, al cual le fueron adicionados algunas construcciones utilizadas en EFDM además de comandos condicionales y de E/S. Originalmente estos comandos fueron definidos en portuguès, pero versiones en otros idiomas pueden ser implementadas via procedimientos de instalación de software. En este artículo adoptaremos una versión con palavras claves en Español. Una descripción detallada de la sintaxis de LBF se encuentra en el apendice y ejemplos sobre su utilización lo mismo que una descripción de su semantica puede ser encontrada en [17].

De manera general, para el modelaje del mundo real son necesarios dos tipos de mecanismos [25]: un mecanismo que permita la descripción de las propiedades estáticas del mundo real, es decir, sus estructuras y las restricciones definidas sobre ellas; y un mecanismo que permita la descripción de las propiedades dinámicas, es decir, las operaciones aplicadas a las estructuras. A continuación describimos como estos mecanismos son implementados en LBF.

3.1 Estructuras

Al igual que DAPLEX, LBF modela las entidades del mundo real y sus propiedades a través de funciones. La declaración de estas funciones es realizada por medio del comando DECLARE. Las funciones pueden tener cero o mas argumentos, siendo que funciones sin argumentos son utilizadas para modelar entidades, mientras que funciones con argumentos son utilizadas para modelar propiedades y relacionamientos. Por ejemplo, considerando la base de datos de la Figura 1,

```
DECLARE ARTICULO () ->> ENTIDAD
```

declara un conjunto de entidades con propiedades comunes (tipo de entidad) llamado ARTICULO y

```
DECLARE TITULO(ARTICULO) -> ALFA(20)

DECLARE DETALLES(ARTICULO) -> ALFA(100)
```

declara sus propiedades (atributos) TITULO como una cadena de 20 caracteres y DETALLES como una cadena de 100 caracteres.

LBF como otros lenguajes basados en el modelo funcional, maneja conjuntos de entidades y conjuntos de valores. Estos ultimos, también llamados de entidades predefinidas, pueden ser clasificados en conjuntos de valores alfanuméricos, numéricos (los cuales pueden tener parte decimal) y lógicos. Los conjuntos de entidades representan objetos del mundo real, pero no son identificadores, es decir, números o nombres que identifican estos objetos externamente, por este motivo no pueden ser utilizados en comandos de E/S. Las funciones son también clasificadas de acuerdo con su resultado, si retornan un conjunto de entidades son llamadas de funciones multivaloradas y si retornan una única entidad son llamadas de funciones simples.

Otra característica de LBF heredada de DAPLEX es la utilización de jerarquías de generalización [21,25]. De esta manera, por ejemplo,

```
DECLARE AUTOR() ->> PERSONA
```

declara un AUTOR como un subtipo de PERSONA, lo que indica que cualquier entidad del tipo AUTOR es también una entidad del tipo PERSONA, y que todas las propiedades y relacionamientos definidos para el tipo PERSONA son auto-

maticamente heredados por el tipo AUTOR.

LBF también permite la definición de funciones derivadas [19] a partir de funciones básicas declaradas a través del comando DECLARE. Las funciones derivadas son definidas por medio del comando DEFINA. Por ejemplo, dada la función

AUTOR(ARTICULO) ->> AUTOR

podemos definir la función derivada (inversa)

DEFINA ARTICULO(AUTOR) ->> INVERSA DE AUTOR(ARTICULO)

que retorna como resultado la lista de articulos publicados por un determinado autor.

Desde el punto de vista estructural, LBF presenta las siguientes diferencias con respecto a DAPLEX:

- Al contrário de la propuesta de Shipman, las funciones simples son por definición consideradas parciales. Esto genera dos ventajas: es posible introducir en la base de datos objetos cuyos datos son incompletos y las funciones inversas pueden ser definidas libremente.

- Shipman permite que argumentos de funciones sean expresiones arbitrárias. La razón para esto radica en el problema de manejar funciones con multiples argumentos. En este caso, si una función utiliza tipos de entidad como argumentos, para algunas combinaciones de estos argumentos la función puede no estar definida. Por ejemplo, en la declaración

DECLARE NUMARTICULOS(AUTOR, PERIODICO) -> NUMERICO(2)

pueden existir pares autor-periódico para los cuales la función no estaría definida, violando el hecho de que las funciones deben ser totales. Como en LBF esta restricción no existe, todos los argumentos deben ser del tipo entidad, lo que simplifica la sintaxis de la definición de funciones.

3.2. Operaciones

Desde el punto de vista de las operaciones, LBF presenta diferencias sensibles en relación a DAPLEX, que son tratadas a continuación.

3.2.1. Operaciones de Selección y Recuperación de Datos

DAPLEX utiliza expresiones y comandos como construcciones básicas para el manejo de datos. Expresiones aparecen siempre dentro de comandos y representan tanto un conjunto de entidades como una sola entidad, lo que permite

clasificarlas como expresiones de conjunto y expresiones simples respectivamente. Las expresiones en general tienen tres caracteristicas: un valor, un papel y un orden. El valor de la expresión es el conjunto de entidades (o la entidad) retornado como resultado de su evaluación. El papel de una expresión está relacionado con el tipo en el cual las entidades del conjunto son interpretadas. El orden está asociado con expresiones de conjunto y representa el orden establecido entre las entidades del conjunto. LBF utiliza la sintaxis de DAPLEX para la especificación de un orden parcial de un conjunto (Vea apéndice).

En lo que respecta a los comandos, existen algunas diferencias entre LBF y DAPLEX. la mas importante es la utilización por parte de LBF de un comando condicional. El comando condicional fue definido para facilitar operaciones (consultas o actualizaciones) dependientes de una condición, evitando asi, que estas sean desmembradas. Por ejemplo, considerando la base de datos de una empresa, la operación de actualización "Aumentar en 25% el salario de todas las personas del Departamento de 'Sistemas y computación' y en 15% el salario de los demás empleados" seria expresada en LBF de la siguiente manera:

```
PARA CADA EMPLEADO

    SI NOMBRE(DEPART(EMPLEADO)) = 'Sistemas y Computación' ENTONCES

       SEA SALARIO(EMPLEADO) = 1.25 * SALARIO(EMPLEADO)

    SINO

       SEA SALARIO(EMPLEADO) = 1.15 * SALARIO(EMPLEADO)

    FIN

FIN
```

El comando SI también es utilizado en la evaluación de expresiones simples y de conjunto. Esta forma de utilización es bastante útil en la definición de funciones derivadas. Por ejemplo,

```
DEFINA POTENCIA (X EM NUMERICO(5), J EM NUMERICO(1)) ->

                    SI J = 0 ENTONCES 1

                    SINO X * POTENCIA(X,J - 1)

                    FIN
```

define la función POTENCIA que calcula el valor de X elevado a la J-ésima potencia.

El primer tipo de comando SI es siempre utilizado dentro de un comando

PARA. Por tanto, ninguna operación puede comenzar con el comando SI. Esta restricción fue impuesta para facilitar la especificación de las operaciones, sin la necesidad de disminuir el poder del lenguaje, ya que cualquier operación que es especificada a partir del comando SI puede igualmente ser definida con la utilización del comando PARA. Una descripción detallada del comando condicional puede ser encontrada en [17].

En lo que se relaciona a los operadores utilizados en las expresiones de conjunto, LBF, al contrario de DAPLEX, provee los operadores UNION, INTERSECCION y DIFERENCIA. Los cuales son utilizados en la unión, intersección y diferencia de expresiones de conjuntos respectivamente. En DAPLEX estos operadores son utilizados exclusivamente en la definición de funciones derivadas. Como operadores sobre expresiones simples LBF utiliza los operadores lógicos Y, O y NO, los operadores aritméticos "+", "-", "*", "/" y RES (módulo), y el operador "&" para la concatenación de cadenas de caracteres.

3.2.2. **Operaciones de Entrada y Salida de Datos**

Para las operaciones de entrada de dados, LBF utiliza el concepto de ENTRADA que equivale a una "entidad externa" cuya función es facilitar el almacenamiento de grandes volumenes de datos a partir de archivos de entrada. Por ejemplo, para modificar la dirección de algunos usuarios de la biblioteca personal (ver Figura 1), bastaria con crear una entidad externa INFORMACION (con sus respectivos atributos) de la siguiente manera,

```
DECLARE INFORMACION() ->> ENTRADA

DECLARE NOMBRE(INFORMACION) -> ALFA(20)

DECLARE DIRECCION(INFORMACION) -> ALFA(30)
```

y ejecutar los comandos

```
PARA CADA INFORMACION

    PARA EL USUARIO TAL QUE

                NOMBRE(USUARIO) = NOMBRE(INFORMACION)

        SEA DIRECCION(USUARIO) = DIRECCION(INFORMACION)

    FIN

FIN
```

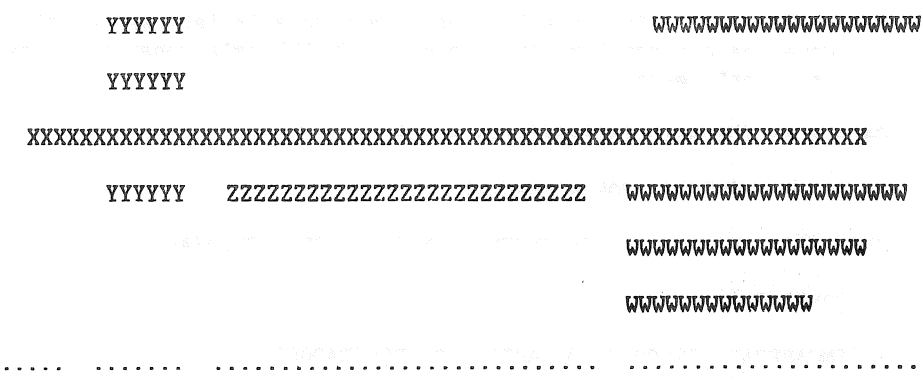
Para la salida de datos, LBF utiliza el comando ESCRIBA (IMPRIMA para emitir reportes por la impresora), que posibilita la generación de reportes simples en un formato predefinido, pero con encabezados, rompimiento por diferentes campos y ordenados acendente o descendentemente. Por ejemplo,

dada la consulta, "Liste los artículos (y sus características) prestados a todos los usuarios que son también autores". En LBF esta consulta es expresada de la siguiente manera:

```
PARA CADA X EN (USUARIO INTERSECCION AUTOR)

     EN ORDEN ACENDENTE POR NOMBRE(X)

     PARA CADA ARTICULO(X) EN ORDEN ACENDENTE POR TITULO(ARTICULO(X))

        CONSIDERE

        ENCABEZADO "REPORTE DE ARTICULOS PRESTADOS".

                   "A USUARIOS QUE SON AUTORES"

        ROMPIMIENTO POR NOMBRE(X), "NOMBRE DEL USUARIO:", NOMBRE(X) 'AP'

        ESCRIBA(TITULO(ARTICULO(X)): "TITULO",

                AUTOR(ARTICULO(X)): "AUTORES",

                PERIODICO(ARTIGO(X)): "PERIODICO DONDE FUE PUBLICADO",

                ASUNTO(ARTICULO(X)): "ASUNTOS RELACIONADOS")

     FIN

FIM
```

Los campos que conforman el reporte siempre seran impresos en la forma de columnas sin importar si la salida pedidad exceda el ancho de la página. Por ejemplo, el reporte anterior sería presentado en el terminal de la siguiente manera:

```
   12 Abr. 1982                                                   Pag.01

                         REPORTE DE ARTICULOS PRESTADOS

                          A USUARIOS QUE SON AUTORES

                                    TITULO

          AUTORES   PERIODICO_DONDE_FUE_PUBLICADO  ASUNTOS_RELACIONADOS

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

          YYYYYY   ZZZZZZZZZZZZZZZZZZZZZZZZZZZZ    WWWWWWWWWWWWWWWWW
```



Una discusión más detallada de los comandos ESCRIBA e IMPRIMA esta por fuera del alcance de este articulo y puede ser encontrada en [17].

3.2.3. Operaciones de Actualización de la Base de Datos

En lo que respecta a las operaciones de actualización, LBF posee las siguientes caracteristicas:

- Debido a que las funciones son por definición parciales, una función simple no necesita ser inicializada en el momento de su creación. Sin embargo, para facilitar la identificación de una entidad externamente, por lo menos una de sus funciones (o un grupo) debe ser inicializada en el momento de su creación.

- En LBF es posible incluir una entidad existente en la base de datos dentro del conjunto de entidades de un determinado tipo. Para esto, es utilizada la misma construcción que es usada para la inclusión de entidades en el conjunto correspondiente al codominio de una función multivalorada. Por ejemplo,

```
INCLUYA USUARIO = (EL AUTOR TAL QUE
                        NOMBRE(AUTOR) = "Shipman")
```

incluye el autor de nombre "Shipman" en el conjunto de usuarios de la biblioteca personal.

- De la misma forma es posible excluir una entidad del conjunto correspondiente a un determinado tipo, lo que resulta en la exclusión de todas las referencias a esta entidad por parte de los conjuntos correspondientes a todos sus subtipos. Por ejemplo,

```
EXCLUYA USUARIO = (EL AUTOR TAL QUE
```

```
NOMBRE(AUTOR) = "Shipman")
```

excluye el autor de nombre "Shipman" del conjunto de usuarios de la biblioteca personal, no obstante el permanece como autor.

- Entidades pueden ser removidas de la base de datos a través del comando REMUEVA. De esta manera,

```
REMUEVA EL AUTOR TAL QUE NOMBRE(AUTOR) = "Shipman"
```

resulta en la completa eliminación del autor de nombre "Shipman", de la base de datos y de todas sus referencias en los conjuntos correspondientes a sus subtipos y supertipos (es decir son eliminadas todas sus instancias).

3.2.4. Operaciones sobre el Esquema

De acuerdo con la propuesta de Shipman, sobre un esquema DAPLEX solamente se permite la adición de nuevas funciones. LBF aprovecha las ideas de Kulkarni [13] y ofrece un nuevo operador, REMOVA, para la eliminación de funciones que el usuario ya no desea mantener en la base de datos. Por ejemplo,

```
REMUEVA DIRECCION(PERSONA)
```

causa la eliminación de la función DIRECCION y de todas las funciones definidas a partir de ella.

3.3. Almacenamiento y Encapsulamiento de Consultas

En el lenguaje LBF, las consultas pueden ser encapsuladas, a través de la palabra clave CONSULTA, e identificadas por un nombre para una referencia posterior. Por ejemplo, la consulta

```
CONSULTA MUJERES:

  PARA CADA PERSONA TAL QUE SEXO(PERSONA) = "F"

      ESCRIBA(NOMBRE(PERSONA))

  FIN

FIN
```

puede ser referenciada y ejecutada en cualquier momento a través de la invocación de su nombre. Además, la consulta puede también ser eliminada con la utilización del comando REMUEVA.

El encapsulamiento es muy útil en el caso de consultas muy usadas y tiene la ventaja de una mayor velocidad de ejecución, ya que el código

generado puede ser almacenado, evitando su compilación cada vez que la consulta es ejecutada.

3.4. Restricciones

La propuesta de Shipman para la especificación de restricciones en DAPLEX incluye el uso de dos construcciones CONSTRAINT y TRIGGER [19]. Estas construcciones, no obstante, no son completas. LBF no soporta estos mecanismos de restricción. Un trabajo posterior deverá introducir la definición de um mecanismo más general para la especificación de restricciones, que pueda ser adicionado posteriormente al lenguaje. No obstante, LBF permite la especificación de restricciones básicas, tales como:

Restricciones de totalidad. Son utilizadas en el caso en que todo objeto perteneciente a un tipo de entidad este siempre asociado a otro objeto en la base de datos. Por ejemplo,

```
DECLARE AUTOR(ARTICULO) ->> AUTOR TOTAL
```

indica que un articulo debe siempre estar relacionado a un autor.

Restricciones sobre la cardinalidad. Las palabras MAXIMO y MINIMO son utilizadas para restringir el número de elementos de los tipos de entidad y de las funciones multivaloradas. Por ejemplo,

```
DECLARE ARTICULOS(USUARIO) ->> ARTICULO MAXIMO 3
```

indica que el número máximo de artículos prestados a un usuario de la biblioteca personal no puede ser mayor que 3.

Restricciones sobre los valores de las funciones. Ciertas funciones pueden ser restringidas de manera que sus valores no puedan ser alterados. Por ejemplo,

```
DECLARE TITULO(ARTICULO) -> ALFA(30) FIJO
```

indica que después de ser atribuido algún valor a este campo (título) por la primera vez este no puede ser alterado nuevamente.

Restricciones sobre la identificación de entidades. En una base de datos los usuarios pueden estar interesados en diferenciar entidades individuales de forma que ellas puedan ser identificadas sin ambiguedad. Por ejemplo,

```
DECLARE NOMBRE(PERSONA) -> ALFA(20) UNICO
```

indica que dos personas no pueden tener el mismo nombre.

4. ESTADO ACTUAL DEL TRABAJO Y ORIENTACIONES PARA ESTUDIOS FUTUROS

LBF está siendo en este momento implementada para microcomputadores compatibles con el IBM PC, teniendo como soporte para el manejo de archivos el sistema Btrieve [23]. Para la implementación del traductor de LBF será utilizado el sistema de implementación de compiladores SIC [1].

Después de esta implementación inicial, nuestro trabajo consistirá en adicionar algunos mecanismos que hasta el momento no fueron considerados, tales como:

Definición de visiones. La propuesta de Shipman relacionada con la definición y actualización de visiones no es completa ya que en ella sólo se trata el caso en que una actualización sobre una visión ocasiona una única actualización sobre la base de datos. EFDM [13], por otro lado, ofrece un mecanismo diferente para definir visiones que no permite visiones actualizables. Nuestro propósito es extender posteriormente el lenguaje LBF, de manera que soporte visiones actualizables considerando el caso de múltiples actualizaciones sobre la base de datos.

El uso de tipos de datos en lenguajes para bases de datos. Un concepto bastante útil para el modelaje del mundo real es el de tipos de datos. Por ejemplo, en PASCAL [10] y EUCLID [14] podemos definir los siguientes tipos de datos

```
TYPE ALTURA = 1...100

TYPE PESO = 1...100
```

Consecuentemente, operaciones sin significado pueden ser efectuadas sobre estos tipos (por ejemplo, sumar pesos con alturas). Por tanto, es necesario que se haga una mejor verificación de tipos para obtener una integridad semántica mayor. Trabajos a este respecto han sido desarrollados en el contexto del modelo relacional [2,21]. Seria interesante extender LBF para que permita diferenciar dos tipos de entidades semanticamente diferentes.

El uso de tipos abstractos de datos. Tipos abstractos de datos han sido ampliamente estudiados en el contexto de lenguajes de programación. Ultimamente, se ha prestado bastante atención a la utilización de abstracción de datos en el contexto del modelaje semántico de datos [5]. En ciertas areas de aplicación, como en la utilización de bases de datos para el soporte a CAD ("Computer Aided Design") existen problemas serios en la representación y operación de objetos. Tipos abstractos de datos (TAD's) pueden ser una alternativa para la creación y manejo de objetos como puntos, líneas y polígonos. Recientemente, algunas implementaciones de TAD's han sido desarrolladas, entre ellas podemos citar el trabajo de Stonebraker en el contexto del sistema INGRES [24]. En este trabajo, fueron definidos TAD's sobre columnas de una relación, es decir, TAD's simples. Un trabajo interesante puede ser extender LBF no sólo para soportar TAD's simples como fue propuesto por Stonebraker [24], sino también para soportar otros tipos más comple-

jos que puedan ser utilizados en aplicaciones de CAD.

5. CONCLUSIONES

Nosotros hemos descrito brevemente los elementos básicos del lenguaje LBF, el cual está basado en el modelo funcional de Shipman. Este modelo no es el modelo semántico más completo pero como ya vimos es um modelo simple de utilizar. Este modelo consigue juntar conceptos como el de orientación a objetos y jerarquías de tipos (generalización) sin perder su simplicidad mientras que otros modelos semánticos como SDM [8] Y RM/T [7], son tan complejos que su implementación se vuelve difícil, además de ser, a veces, poco adecuados para la mayoría de usuarios. Modelos del tipo DAPLEX pueden facilitar, por tanto, el acceso de un mayor número de personas a la tecnología de bases de datos.

6. BIBLIOGRAFIA

[1] BIGONHA, M.A.S., "SIC : Sistema de Implementação de Compiladores". Belo Horizonte,Departamento de Ciência da Computação (DCC) ICEX-UFMG, Junho, 1985. (Tesis de Maestrado)

[2] BRODIE, M.L. "The Aplication of Data Types to Database Semantic Integrity". **Information Systems**, Pergamon Press , 5(4):287-296, 1980.

[3] BRODIE, M.L. & SILVA, A. "Active and Passive Component Modelling". In: OLLE, T. W. et al (eds.). "**Information Systems Design Methodologies: A Comparative Review**". North-Holland, Amsterdam, 1982, p. 41-91.

[4] BRODIE, M.L. "On the Development of Data Models". In: BRODIE, M.L., MYLOPOULOS, J. & SCHMIDT, J.W. (eds.). "**On Conceptual Modelling. Perspectives from Artificial Inteligence, Databases, and Programming Languages**". New York, Springer Verlag, 1984, p.21-47.

[5] BRODIE, M.L. & RIDJANOVIC, D. "On the Design and Specification of Database Transactions". In: BRODIE, M.L., MYLOPOULOS, J. & SCHMIDT, J.W. (eds.). "**On Conceptual Modelling. Perspectives from Artificial Inteligence, Databases, and Programming Languages**". New York, Springer Verlag, 1984, p.277-306.

[6] BUNEMAN, P. & FRANKEL, R.E. "FQL - A Functional Query Language". In: **PROCEEDING OF ACM SIGMOD CONF. ON MANAGEMENT OF DATA**, Boston, Mass., 1979, p. 52-58.

[7] CODD, E.F. "Extending the Relational Model of Data to Capture more Meaning". **ACM Tranactions on Database Systems**, New York, 4(4):397-434, 1979.

[8] HAMMER, M. & MCLEOD, D. "Database Description with SDM: a Modelling Mechanism for Database Aplications". **ACM Transactions on Database**

Systems, New York, 6(3):351-386, 1981.

[9] HOUSEL, B.C. , WADDLE, V. & YAO, S.B. "The Functional Dependency Model for Logical Database Design". In: **PROCEEDINGS OF INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES**, 5, Rio de Janeiro, Brazil, Out. 1979, p.194-208.

[10] JENSEN, K. & WIRTH, N. "PASCAL User Manual and Report", 2nd Ed., Lectures notes in Computer Science, Berlim, Springer-Verlag, 1974, p.18.

[11] KATZ, R.H. & WONG, E. "Resolving Conflicts in Global Storage Design through Replication". **ACM Transactions on Database Systems**, New York, 8(1):110-135, 1983.

[12] KERSHBERG, L. & PACHECO, J.E.S. **"A Functional Data Base Model"**. Rio de Janeiro, Brazil, Pontificia Universidade Católica, Fev. 1976. (Monograph in Computer Science 2/76).

[13] KULKARNI, G. K. **"Evaluation of functional data models for database design and use"**. Edinburgh, University of Edinburgh, 1983. 153p. (Tesis de Ph.D.).

[14] LAMPSON, B.W. & et al. "Report on the Programming Language EUCLID", **SIGPLAN Notices**, 12(2), 1977.

[15] McLEOD, D. & SMITH, J.M. "Abstractions in Databases". In: BRODIE, M.L. & ZILLES, S.N. (eds.) PROC. WORKSHOP ON DATA ABSTRACTION, DATABASIS AND CONCEPTUAL MODELLING. **SIGPLAN Notices**, New York, 16(1):19-25, 1981.

[16] ORMAN, L. "Functions in Information Systems". **Data Base**, 16(3):10-13, 1985.

[17] PEREZ, J.J.,LAENDER, A.H.F. & BIGONHA, R.S. **"Especificação da Linguagem para Banco de Dados Funcionais (LBF)"**. (Reporte técnico en elaboración).

[18] SCHIEL, U. "An Abstract Introduction to the Temporal Hierarchic Data Model (THM)", In: **PROCEEDINGS OF INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASE**, 9, Florence, Italy, 1983.

[19] SHIPMAN, D. W. "The Functional Data Model and the Data Language DAPLEX". **ACM Transactions on Database Systems**, New York, 6(1): 140-173, 1981.

[20] SIBLEY, E.H. & KERSCHBERG, L. "Data Architecture and Data Model Considerations". In: **AFIPS CONFERENCE PROCEEDINGS, NATIONAL COMPUTER CONFERENCE**, Montvale, N. J., AFIPS Press, 1977, V.46, pp.85-96.

[21] SMITH, J. M. & SMITH D.C.P. "Database Abstractions: Aggregation and Generalization". **ACM Transactions on Database Systems**, New York, 2(2):105-133, 1977.

[22] SMITH, J.M. et al. "Multibase - Integrating Heterogeneous Distributed Database Systems". In: **AFIPS CONFERENCE PROCEEDINGS, NATIONAL COMPUTER CONFERENCE**, Montvale, N.J., AFIPS Press, 1981, V.50, p.487-499.

[23] SOFTCRAFT, Inc. **"Btrieve version 3.0 user's guide"**. Austin, Texas, 1984.

[24] STONEBRAKER M. "Adding Semantic Knowledge to a Relational Database System". In: BRODIE, M.L., MYLOPOULOS, J. & SCHMIDT, J.W. (eds.). **"On Conceptual Modelling. Perspectives from Artificial Inteligence, Databases, and Programming Languages"**. New York, Springer Verlag, 1984, p.333-353.

[25] TSCICHRITZIS, D. & LOCHOVSKY, F.H. **"Data Models"**. Prentice-Hall, Englewood Cliffs, N.J., 1982, 381p.

[26] WIRTH, N. "What can we do about the unnecesary diversity of notation for sintactic definition?". **Communications of ACM**, 20(11):822-823, 1977.

[27] WONG, E. & KATZ, R.H., "Logical Design and Schema Conversion for Relational and DBTG Databases". In : CHEN P. P.(ed.) **"Entity-Relationship Approach to Systems Analysis and Design"**. Amsterdam, North-Holland, 1980, p.344-383.

APENDICE

DEFINICION SINTACTICA DE LBF.

1. Notación para la descripción sintactica

Para describir la sintaxis de LBF usaremos el siguiente formalismo [26]: los símbolos no terminales de la gramática seran descritos en letras minúsculas y los terminales seran siempre colocados entre comillas. Cada producción tendrá la forma

S = E

donde S representa un símbolo no terminal y E las alternativas que definen S. La expresión E tiene la forma

T1 | T2 | ... | Tn (n > 0)

donde Ti (que llamaremos término i) tiene la forma

F1 F2 ... Fn (n > 0)

donde cada Fi (que llamaremos factor i) puede ser:

(a) un símbolo terminal
(b) un símbolo no terminal
(c) {T}
(d) [T]
(e) la sentencia vacía.

Por tanto cada término pude generar una concatenación de factores que a su vez pueden generar los casos anteriores. Un término entre llaves representa una secuencia de ese mismo término, incluyendo la sentencia vacía. Un término entre corchetes representa ese mismo término o la sentencia vacía. Pares de paréntesis pueden ser usados libremente para indicar agrupamientos.

2. Sintaxis de LBF

```
comando = declarativo | imperativo
        |"CONSULTA" idconsulta ":" {imperativo} imperativo
         "FIN" [idconsulta]
        |"EJECUTE" idconsulta idarchivo
        |"REMUEVA" (especfun|idconsulta)
        |idconsulta

declarativo = "DECLARE" especfun ("->"|"->>")
```

```
                    (idtipo[orden] {restr}
                     | "ENTIDAD"
                     | "ENTRADA")
            |"DEFINA" especfun ("->"|"->>")
                    (expr|
                     "INVERSA DE" especfun|
                     "TRANSITIVA DE" expr|
                     "(" tupla ")"
                    ) [orden] {restr}

restr = "TOTAL"|"MAXIMO" int|"MINIMO" int|"FIJO"|"UNICO"

tipoprim  =  "ALFA" "(" int ")"| "NUMERICO" "(" int[","int]  ")|
             "LOGICO"

expr = expr_conjunto|expr_simple

tupla = expr {"," expr}

especfun = idfun "(" [idtipo {"," idtipo}] ")".

expr_conjunto = "SI" predicado "ENTONCES" expr_conjunto
                ["SINO" expr_conjunto] "FIN"
  | llamfunmv | idtipo | "{" [simple { "," simple } ] "}" | tipoprim
  |"("expr_conjunto {("UNION"|"INTERSECCION"|"DIFERENCIA") expr_conjunto}")"
  | identificador "EN" expr_conjunto | expr_conjunto "COMO" idtipo
  | expr_conjunto "TAL QUE" predicado
  | expr_conjunto comp (expr_simple | cuant expr_conjunto)

expr_simple = termolog [ "O" termolog ]

termolog = factorlog [ "Y" factorlog]

factorlog = [ "NO" ] exprlog

exprlog = expraritm [ comp exprearitm]

expraritm = [prefijo] termo [opsuma termo]

termo = factor [opmult factor]

factor = exprentid ["COMO" idtipo]

exprentid = constante | idvar | llamfuns | llamag | predicado
     | ("EL"|"LA") expr_conjunto | "UN NUEVO" idtipo
     | ("EL"|"LA") expr_conjunto ("QUE PRECEDE" | "QUE SIGUE") expr_simple
     | "SI" predicado "ENTONCES" expr_simple ["SINO" expr_simple] "FIN"
     | "(" expr_simple ")"

llamfuns = llamfun
```

```
llamfunmv = llamfun

llamfun = idfun "(" [tupla] ")"

llamag = ("CARDINAL"¦"MAXIMO") "("expr_conjunto")"
       ¦ ("TOTAL"¦"MEDIA") "("["SOBRE"] expr_simple ")"

predicado = lógico
          ¦ "PARA" (expr_simple ¦ cuant expr_conjunto) predicado
          ¦ expr_simple comp cuant expr_conjunto
          ¦ cuant expr_conjunto comp (expr_simple¦cuant expr_conjunto)
          ¦ cuant expr_conjunto ("EXISTE" ¦ "EXISTEN")

comp = ">" ¦ "<" ¦ "=" ¦ ">=" ¦ "<=" ¦ "<>"

cuant = "ALGUN" ¦ "TODO" ¦ "NINGUN"
       ¦ (("AL" "MENOS" ¦ "ALO" "SUMO")) ¦ "EXACTAMENTE") entero

entero = simple

lógico = simple

constante = ent ¦ cad ¦ log ¦ num

ent = digito{digito}

num = [ent] "." int

cad = """" letra{letra}letra """"

log = "VERDADERO" ¦ "FALSO"

imperativo = "PARA CADA" expr_conjunto[orden] cuerpo "FIN"
           ¦"PARA" simple cuerpo "FIN"
           ¦actualización
           ¦impresión

cuerpo = (imperativo¦selección) {imperativo¦selección}

selección = "SI" predicado "ENTONCES" cuerpo
              ["SINO" cuerpo] "FIN"

orden = "EN ORDEN" {[("ACENDENTE" ¦ "DECENDENTE")] "POR" expr_simple}.

actualización = "SEA" (llamfuns¦llamfummv¦idvar) "=" expr
           ¦"INCLUYA" (llamfunmv¦idtipo) "=" expr
           ¦"EXCLUYA" (llamfunmv¦idtipo) "=" expr
           ¦"REMUEVA" expr_simple
           ¦"INSERTE" llamfunmv "=" (expr_simple ¦ expr_conjunto[orden])
                                "QUE" ("PRECEDE" ¦ "SIGUE") expr_simple
```

```
impresión = ["CONSIDERE" [titulo] [rompimiento] [total]]
     ("ESCRIBA"¦"IMPRIMA") "("expr ":" cad {"," expr ":" cad")" idarchivo

titulo = "ENCABEZADO" texto {"." texto}

texto = (idvar ¦ cad ¦ expr_simple) {"," (idvar ¦ cad ¦ expr_simple)}

rompimiento = "ROMPIMIENTO POR" expr_simple ["," texto "'"opcionesq"'"]
                    {"," expr_simple ["," texto "'"opciones"'"]}

opcionesq = P : causa  un  salto de página después de la impresión de  los
                datos asociados al rompimiento en curso.

            A : causa que el texto sea emitido antes de los  detalles.  En
                el caso de que la opción no sea espicificada el texto será
                colocado después de los detalles.

total = "TOTAL POR" expr {"," expr } ["GRAN TOTAL" texto]

idvar = identificador

idtipo = identificador

idfun = identificador

idconsulta = identificador

idarchivo = identificador

prefijo = "+" ¦ "-"

sumop = "+" ¦ "-" ¦ "&"

mulop = "*" ¦ "/" ¦ "RES"

identificador = letra{caracter_de_id}

caracter_de_id = letra
                 ¦ digito
                 ¦ "_"

letra = "A" ¦ "a" ¦ "B" ¦ "b" ¦ ... ¦ "Z" ¦ "z"

digito = "0" ¦ "1" ¦ ... ¦ "9"
```

AGRADECIMIENTOS

Este trabajo está siendo desarrollado con el apoyo finaciero de la CAPES y del CNPQ (procesos número 302107/81-CC y 402170/85-CC).

GERENCIAMENTO DE PROBLEMAS EM REDES

Liane Margarida Rockenbach Tarouco
Universidade Federal do Rio Grande do Sul
Porto Alegre - *Brasil*

1. O GERENCIAMENTO DE PROBLEMAS EM REDES DE COMPUTADORES

O gerenciamento dos problemas é o esforço para identificar, rastrear e resolver situações envolvendo falha física de algum compoennete de hardware ou de alguma facilidade de transmissão, mau funcionamento de software ou aspectos de treinamento que afetem o funcionamento adequado da rede. Os problemas que ocorrem em redes de computadores podem ser classificados de varias maneiras:

intermitentes x constantes
internos x externos
software x hardware x ambientais

Os problemas podem ser ou não passíveis de repetição, quando se consegue criar as condições idênticas em que ocorreu anteriormente. A detecção de problemas é um mecanismo usualmente disparado devido à ocorrência de um erro na operação de algum componente do sistema.

Nos esquemas tradicionais, tem-se um operador controlador da rede, que, de uma console, de onde monitora a rede, mantem-se ao par das suas condições, trabalhando com vistas a efetuar o diagnóstico sobre os problemas da rede. O diagnóstico de um problema pode ser sub-dividido em três níveis, conforme sugerido por Lenox /LEN 84/. O primeiro nível de diagnóstico, ocorre quando chega o relato do problema. Notificações de problemas podem chegar, por telefone, telex ou alarme pelo sistema. O operador interage então com o computador, executando uma séries de passos visando a determinação da causa do problema. Informa ao sistema, os dados de identificação do equipamento envolvido, abrindo uma ficha de ocorrência. É recuperada toda a informação existente no banco de dados, pertinente ao problema: modificações na topologia de rede ou problemas anteriores afetando aquele dispositivo. Esta é a fase de diagnóstico do problema, mediante análise de toda a informação relevante disponível.

Se o problema pode ser imediatamente corrigido, o evento é classificado como um incidente operacional. Senão, é classificado como um problema e encaminhado para uma área especializada. Neste caso, o operador envia a ficha de ocorrência a um técnico encarregado de reparar o problema (integrante dos quadros da própria empresa ou do fornecedor do equipamento envolvido). Costuma ser assinalado na ficha de ocorrência, um código indicativo do impacto do problema sobre o usuário. Isto permite à área especializada determinar prioridades de atendimento, se vários problemas carecem de atenção simultaneamente. Na área especializada ocorre um segundo nível de diagnóstico. São feitos testes sobre os circuitos e outros equipamentos envolvidos (modems, multiplexadores, etc.) sendo acionado o pessoal de manutenção apropriado. Os equipamentos a serem testados podem estar remotos, assim, urge dispor de facilidade para teste remotamente comandado. Após o problema ser diagnosticado, ação corretiva deve ser tomada, de modo que o sistema possa voltar a funcionar corretamente.

O terceiro nível de diagnóstico envolve a atuação de programadores e analistas, sendo concernente ao funcionamento dos programas de aplicação e software básico, quando o problema é derivado de algum bug de software ou dimensionamento inadequado.

Após a solução do problema, deve ser registrada no banco de dados de apoio à gerência de rede, informação concernente a causa e conserto do problema, além outras informações tais como: tempo e outros recursos dispendidos para a solução, reconfigurações efetuadas e soluções paliativas, pendentes de providencias futuras.

Enquanto o problema não foi solucionado, existem três alternativas a seguir, para assegurar a continuidade do serviço:

- uso de equipamentos redundante
- encaminhamento a outra fonte de atendimento (outro terminal, outro concentrador, outra forma de comunicação, outro computador etc.)
- reconfiguração

Os custos derivados da perda da capacidade de atendimento podem ser altos, bem como os custos inerentes a manutenção de facilidades adicionais, redundantes, para uso em caso de falha. Por outro lado, diagnóstico incorreto ou tentativa de retomar o uso do componente com problemas, antes que os concertos adequados sejam efetivados somente aumenta os custos, aumentando o tempo de paralização e a frustação dos técnicos e dos usuários envolvidos.

Em vista de tudo isso, deseja-se evidenciar a necessidade de um esquema compreensivo de gerenciamento e uma arquitetura projetada para reduzir os custos diretos do processo de restauração da operacionalidade da rede, bem como para prover a informação e flexibilidade neces-

sária, pois, conforme Hart /HAR 83/, localizar as causas de problemas numa rede não é uma tarefa fácil.

2. DIFICULDADES ENCONTRADAS NO GERENCIAMENTO DE PROBLEMAS EM REDES

O contínuo crescimento em número e diversidade dos componentes das redes de computadores tornou a atividade de gerenciamento de rede muito complexo. Isto se agrava quando estão envolvidos muitos fornecedores. Já foi afirmado que "o número de problemas numa rede é proporcional ao quadrado do número de fornecedores envolvidos". O isolamento e teste dos problemas das redes tornou-se muito difícil devido a varias causas:

- Muitos níveis de pessoal envolvido técnicos de manutenção, operadores controladores de rede, gerentes de sistemas de informações e gerentes de comunicações.

- Diversidade de formas de controle e monitoração. Embora os produtos envolvidos na rede tornem-se gradativamente mais complexos, cada fornecedor oferece ferramentas de controle de rede próprias, para monitorar seus produtos.

- Não somente equipamentos ou linhas podem evidenciar problemas, mas, em princípio qualquer entidade, inclusive módulos de software ou mesmo níveis inteiros, no conceito do modelo OSI.

- O centro de diagnóstico e controle das redes não tem sempre acesso direto a todos os componentes numa rede distribuida. Logo, precisa-se de um protocolo para interação entre eles.

3. SOLUÇÕES PARA A GERÊNCIA DA REDE

Para apoiar a atividade de gerenciamento de uma rede de computadores, deve ser montado um sub-sistema de apoio à atividade de gerência de rede com funções de:

- Prover registro de problemas, de forma integrada e automatizada, conforme exemplificado por Mines /MIN 84/. Isto significa que quaisquer anormalidades no comportamento de algum componente da rede, serão registradas local ou centralizadamente. No primeiro caso, em algum momento posterior, tais informações deverão ser repassadas, em forma bruta ou agregada, para um gerenciador central, onde serão processadas de modo a resultar em informação adequada para apoiar a tomada de decisão do gerente da rede.

- Questionar outras entidades buscando informações sobre ocorrências e eventos indicadores de possíveis problemas, pois nem sempre as informações sobre os problemas acontecidos virão

expontaneamente para o ponto onde deverão ser processadas em conjunto para resultar em informação útil à gerência da rede. Para este fim, são necessários, protocolos para transmissão de tais informações, conforme referido por Robinson /ROB 84/.

- Detectar tendências no comportamento de componentes da rede. Uma análise das informações disponíveis sobre os problemas acontecidos na rede pode conduzir a uma previsão sobre a possível repetição ou intensificação de alguns dos problemas. Se, para apoiar esta análise, tivermos um sistema especialistas, tal como sistema ACE, descrito por Stolfo /STO 82/, esta tarefa fica muito mais facilitada.

- Registrar alterações e/ou consertos na rede. Em virtude de mau funcionamento de algum componente da rede, é necessário, muitas vezes a substituição de equipamentos ou até mesmo o redirecionamento de parte do serviço efetuado na rede para outros pontos, ou usando outras rotas. Tão logo os componentes originais se tornem operacionais, é preciso restaurar a rede à sua plena operacionalidade desfazendo as alterações comandadas no periodo de operação paliativa. Por outro lado, após concluir a solução de algum problema, isto deve ser registrado para que, posteriormente, se possa avaliar MTBF e MTTR (Mean Time Between Failures e Mean Time To Repair) dos diversos integrantes da rede.

A gerência da rede pode ser centralizada, (localizada num só compoennete da rede) ou distribuída. Deve ser montado um banco de dados que contenha informações adequadas, tais como as necessárias para o isolamento de problemas. Este banco de dados pode ser tambem distribuido. Neste caso os dados podem ser dispostos de maneira particionada, totalmente redundante ou parcialamente redundante. No último caso, algumas informações, normalmente mais detalhadas, ficariam nas bases locais, enquanto dados de interesse mais global ou valores agregados, ficariam num sub-sistema a parte, responsável pelo controle das atividades de gerência. Caso este necessitasse de dados armazenados nas bases locais, poderia interagir com os componentes onde as mesmas se encontrassem, para requisitar tais dados.

Como, se pode deduzir de tudo o que foi referido até aqui, o mecanismo de apoio à atividade de gerenciamento da rede constitui a ferramenta mais importante para rastrear e resolver problemas. O projeto de um esquema deste tipo deverá, contudo, incorporar as seguintes características:

- O gerenciamento da rede deverá ser parte integral da mesma

- Deverão ser permitidos múltiplos pontos de acesso ao gerenciamento da rede (estações de controle)

- A informação sobre a rede, bem como as estatísticas da sua atividade (normal ou dos problemas), incluindo os dados de eventos, deverão poder ser apresentados em forma facilmente compreensivel, talvez usando gráficos, conforme defendido por Currie /CUR 82/.

- Um esquema de tratamento prioritário deve permitir que as mensagens de controle da rede precedam outros tipos de tráfego

- Devem existir mecanismos de segurança que limitem o acesso à rede e detectem uso não autorizado.

- As funções de gerenciamento de rede devem operar independentemente do meio de transmissão

- Deverá existir um banco de dados contendo informação sobre todos os componentes da rede e de seus usuários.

- Alterações na rede deverão poder ser efetivadas de forma flexível e simples

A integração das funções de gerenciamento na rede é importante e não pode ser acomodada pela simples adição de equipamento extra. Nas atuais arquiteturas de rede, em que é usada uma estratégia de estratificação de funções em níveis, o gerenciamento deve também ser parte das funções inerentes a cada nível. Por exemplo, testes que envolvam o interface físico do equipamento (tal como o comando de teste por lopp-back do conector modem-equipamento de processamento de dados) devem estar embutidos no nível físico, devendo ser possível seu acionamento, de alguma forma, controlado pelo sistema gerenciador da rede. Isto implica na existência de um protocolo, para interação entre os componentes de gerenciamento da rede, que inclua a possibilidade de envio de ordens para efetivação de alguma rotina de teste de componentes ou de mesmo para uma leitura no registro local que contem todos os eventos ocorridos com os componentes diretamenet conectados com aquele ponto.

Uma importante parte do processo baseaia-se na apropriação de informação sobre a rede, sendo as mais importantes aquelas relativas a erros, falhas e outras condições problemáticas. Tais dados devem ser armazenados em forma bruta mas também é importante ter valores aceitáveis como limiar de tolerância, que, quando ultrapassado, determinam uma sinalização ao operador ou início de uma ação corretiva. Tais limites não são necessariamente absolutos, tal como o número de erros num circuito por unidade de tempo, sendo necessário dispor de estatísticas de erros em função do tráfego existente. Um determindado limiar pode ser aceitável numa situação de carga leve na rede mas intolerável numa outra situação, de carga mais intensa, onde o número de retransmissões faria com que o tráfego total excedesse a capacidade do enlace, afetando seriamente o tempo de resposta.

Quando a rede encontra-se em situação de operação em forma degradada, pode ser necessário tomar decisões quanto à suspensão de certas atividades, tendo em vista a capacidade diminuida da rede e a importância e grau de urgência dos serviços que devem continuar a ser prestados. Esta importância e grau de prioridade pode variar, segundo a hora do dia ou o local. Daí a importância de um sistema de suporte à decisão para apoiar a atividade de manutenção, conforme descrito por Chitke /CHI 84/.

O gerenciamento da rede deve implicar na existência de um banco de dados contendo informações completas sobre todos os elementos da mesma (linhas, modems, processodores de rede, terminais, computadores, software, ect.). Para cada um dos itens o operador da rede deve ser capaz de acessar informações completas tais como: proprietário, localização, custo operacional, arrendatário, número serial, identificação do circuito, uso normal, etc, se o item fosse um modem. O acesso deve ser facilitado por meio de linguagens de alto nível e protocolos para adicionar e manipular dados neste banco de dados, bem como acessá-los para o cumprimento de funções de gerencimento de rede.

Na medida em que a complexidade da rede aumenta, um vídeo gráfico (preferencialmente colorido) deve oferecer possibilidade de mostrar a rede toda, indicando a natureza e o local das falhas e problemas. Condições críticas devem ser assinaladas por meio de sinais de alerta. Na medida em que forem sendo efetivadas reconfigurações, a rede resultante deve ser visualizada graficamente. Uma facilidade de aproximação ("zoom") deve permitir ao operador visualizar algum ponto, com grau de detalhamento maior.

Um componente também importante no esquema de apoio ao gerenciamento de rede é a orientação no que tange aos passos ou etapas que devem ser cumpridos. Cada nível de diagnóstico tem um roteiro próprio. O funcionamento de alguns roteiros implica no uso de temporizadores internos que emitem um alerta se certos limiares de tempo são atingidos sem que o problema seja solucionado.

A maioria dos esquemas de gerenciamento de rede existentes atualmente são deficientes, no que tange as necessidades aqui discutidas, especialmente quando é utilizado equipamento de múltiplos vendedores. É surpreendente constatar que redes de grande porte têm, as vezes, uma estrutura de gerencimento de problemas manual baseada em papel. Isto funciona quando a rede é pequena, mas a medida em que ela cresce, o processo torna-se incapaz de sequer registrar o universo dos incidentes, nesta forma manual.

O melhor caminho, para uma solução integral ao gerenciamento de problemas em redes de computadores, seria um esquema de gerenciamento de rede coerentemente estruturado, como um elemento de controle integral, capaz de interagir com os mais diversos pontos da estrutura, recebendo expontaneamente alarmas ou outros indicadores de problema ou buscando tais indícios mediante interação.

Uma solução assim genérica e integrada auxiliaria os usuários a evitar os altos custos de um pacote gerenciador "customizado" e facilitaria uma manutenção econômica, o futuro crescimento e evolução da rede, bem como o suporte técnico para sua operação. No sentido de definição de uma solução com estas características está sendo estudada pelo ISO/TC 97/SC 21/WG 4, uma proposta de padronização para o serviço de gerenciamento de rede, que será comentado a seguir.

4. A PROPOSTA DA ISO PARA GERENCIAMENTO DE REDE

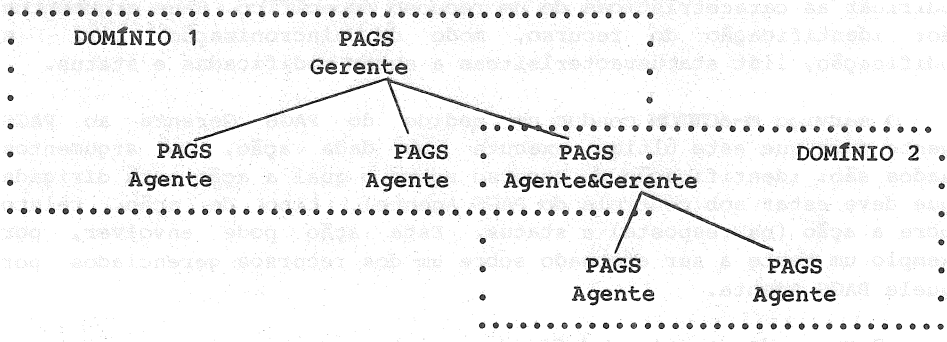
O modêlo de referência OSI introduz o conceito de gerenciamento no ambiente OSI e identifica uma série de atividades gerenciais, concernentes à ativação/desativação de conexões, monitoração e controle de erro. A partir disto foi elaborada, pelo grupo de trabalho ISO/TC 97/SC 21/WG 4, uma extensão do modêlo de referência básico OSI incluindo conceitos e terminologia para descrever o modêlo de gerenciamento no contexto OSI, seus objetivos e serviços e uma descrição de seus componentes /ISO 85/.

O gerenciamento no contexto OSI identifica três categorias de informação gerencial a ser intercambiada: notificação de eventos, transferência de informação e controle. Para transferir estas informações são usados os Serviços de Informação Gerencial (SIG).

Os Serviços de Informação Gerencial (SIG) são definidos como uma série de serviços de nível de aplicação, usados para intercambiar informação gerencial e de controle. Os serviços de Informação Gerencial são prestados por um Processo de Aplicação de Gerenciamento de Sistema (PAGS ou SMAP-System Management Application Process). Existirá ao menos um PAGS em cada sistema envolvido com a transferência de informação gerencial. Este PAGS comunica-se com outros PAGSs ou com um processo de aplicação comum para transferência das informações de controle necessárias, tais com o relato da ocorrência de um particular erro, o pedido para o estabelecimento de um particular parâmetro de configuração etc.

Como nem todos os sistemas terão plena capacidade para apoiar integralmente as atividades de gerenciamento, foi definido o conceito de PAGS "Gerente" e PAGS "Agente" em que o primeiro possui todas as capacidades previstas e o segundo tipo apenas um conjunto limitado delas. Um PAGS gerente controlando um conjunto de PAGS "Agentes" constititui um DOMINIO. Um PAGS pode ser membro de mais de um dominio. Os dominios podem ter uma estrutura hierárquica em que um PAGS "Gerente" num domínio pode ser "Agente" em outro domínio, conforme exemplificado por Roos /ROO 86/.

Figura 1: Hierarquia de domínios



Um PAGS interage com outro PAGS através de uma Entidade de Aplicação de Gerenciamento de Sistemas (EAGS). Tais entidades provem uma série de serviços, classificados em três categorias:

-ESCA:Elementos de Serviço Comuns à Aplicação
-ESEA:Elementos de Aplicação Específicos à Aplicação
-ESIG:Elementos de Serviço de Informação Gerencial

Um PAGS ou um processo de aplicação comum utiliza os ESIG para o intercâmbio de informação de gerenciamento da rede. Foram definidos dois tipos de ESIG: os comuns e os específicos. Os comuns (c-ESIG) dizem respeito a um conjunto de serviços generalizados para o intercâmbio de informação gerencial; os específicos (e-ESIG) são particulares aos vários tipos de informação gerencial intercambiada (contabilização, problemas, configuração, performance e segurança). Os c-ESIG atualmente definidos pela ISO são os seguintes:

M-EVENT-REPORT
M-GET-ATTRIBUTES
M-SET-CHARACTERISTICS
M-ACTION

O serviço **M-EVENT-REPORT** é usado para notificar o PAGS "Gerente" apropriado sobre a ocorrência de um evento. A notificação de ocorrência de evento pode ser confirmada ou não. Os argumentos deste elemento de serviço incluem: identificação do evento, do recurso, tipo de evento, seu horário de ocorrência, informações adicionais sobre o evento, confirmação e status.

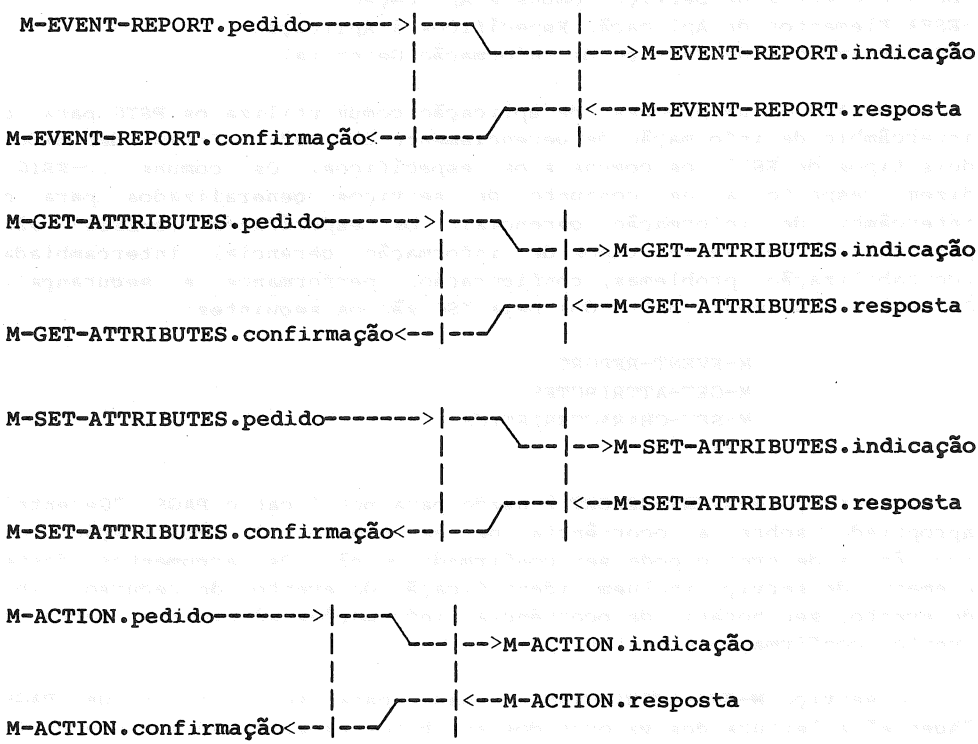
O serviço **M-GET-ATTRIBUTES** é usado para solicitar a um PAGS "Agente" a leitura dos valores dos atributos especificados e seu envio ao PAGS "Gerente". Os parâmetros utilizados são:identificação do recurso, sincronização para envio dos valores, lista de atributos desejados, seu valor (na resposta) e status.

O serviço **M-SET_CHARACTERISTICS** permite a um PAGS "Gerente" modificar as caracetrísticas de um recurso específico. Seus argumentos são: identificação do recurso, modo de sincronização para a modificação, list statusracterísitcas a serem modificadas e status.

O serviço **M-ACTION** conduz um pedido do PAGS Gerente ao PAGS Agente para que este último execute uma dada ação. Os argumentos usados são: identificação do recurso sobre o qual a ação será dirigida (que deve estar sob controle do PAGS Agente), tipo de ação, relato sobre a ação (na resposta) e status. Esta ação pode envolver, por exemplo um teste a ser efetuado sobre um dos recursos gerenciados por aquele PAGS Agente.

Para cada serviço definido existem as quatro primitvas, definidas no modelo OSI (pedido, indicação, resposta e confirmação). A figura 2 mostra as sequências em que estas primitivas são executadas.

Figura 2:Sequência de primitivas c-ESIG



As primitivas de serviço específicas a cada tipo de informação de gerenciamento de rede intercambiada (e-ESIG) deve prover, no caso de gerenciamento de problemas, que é o foco deste trabalho, opções para interação entre os PAGS Agente e Gerente visando relatar problemas. Ora, problema é qualquer coisa que faça com o sistema opere de modo anormal. Além das classificações referidas na seção 1, deste trabalho, podemos ainda agrupar os problemas por nível, segundo o modêlo OSI, isto é, que tipo de entidade pode experimentar qual tipo de problema.

Um mapeamento inicial poderia ser o da figura 3.

Figura 3: Níveis OSI e problemas inerentes

	NÍVEIS						
PROBLEMAS	FÍSICO	ENLACE	REDE	TRANSP.	SESSÃO	APRES.	APLIC.
Rejeição sem explicação		X	X	X	X	X	
Desconexão anormal	X	X	X	X	X		X
Endereço desconhecido			X	X	X		X
Endereço inatingível	X	X	X	X	X		
Dessequenciação		X	X	X	X		
Congestionamento			X	X	X		
Versão do protocolo inaceitável		X*		X	X	X	
Classe de serviço não disponível			X	X	X	X	
Capacidade não disponível			X	X	X	X	X
Formato inválido		X	X	X	X	X	X
Mensagem muito longa		X		X		X	
Erro de transmissão		X		X			
Erro de protocolo		X	X	X	X		
Acesso barrado (password)			X	X	X		X

Obs.:(*) em redes locais

Para detectar cada um destes problemas, já existem mecanismos, previstos nos protocolos utilizados pelas entidades de cada nível. Assim, cada entidade armazenaria, na base de informações local, o registro da ocorrência de cada um destes problemas, bem como dados para identificar o(s) componente(s) envolvidos, além da hora de ocorrência do problema. Tais dados, seria passados, pelo PAGS Agente, em forma bruta, ou agregagos, para o PAGS Gerente de cada domínio, segundo protocolos ainda não completamente definidos.

O PAGS Gerente, processaria tais dados, apoiado por um sistema de suporte, tal como um sistema especialista, oferecendo ao Gerente da Rede a informação necessária ao correto desempenho de sua função.

5. CONCLUSÕES

Embora se possa descrever funcionalmente o conjunto de mecanismosa que devem existir para prover o Gerente de uma Rede de Computadores com ferramentas adequadas à sua atividade, existe ainda muito a ser feito. A caracterização mais detalhada dos módulos funcionais e da base de dados com informações de gerenciamento da rede é algo ainda por completar. O projeto e implantação de um sistema especialista para manuseio de toda esta informação, por outro lado é tambem uma tarefa por fazer.

Contudo, acredita-se que esta área constitui atualmente um dos mais fascinantes campos de pesquisa em Comunicação de Dados e deverá experimentar um grande e acelerado progresso, pois as redes de computadores atualmente existentes demandam soluções capazes de tornar seu correto funcionamento não um golpe de sorte mas o resultado de um trabalho sistemático e bem fundamentado em informações apropriadas.

BIBLIOGRAFIA:

/CHI 84/ Chikte, S. et allii. A decision support system for transmission facility maintenance. Proccedings IEEE International Conference on Communications. Amsterdan, May, 1984.

/CUR 82/ Currie, W. Network status display system. Computer Communications, vol. 5 nº 1, Feb., 1984.

/HAR 83/ Hart, Larry. For network managers, finding faults is no easy task. Data Communications, September, 1983.

/ISO 85/ ISO. OSI - Management Information Service Definition, ISO/TC 97/SC 21/WG 4. Philadelphia, Nov, 1985.

/LEN 84/ Lenox, Tim and Dean, Rod. Improve problem management step by step. Data Communications, June, 1984.

/MIN 84/ Mines, J. et allii. Integrating performance monitoring with other alarm informations to enhance digital transmission systems. Proceedings IEEE International Conference on Communications. Amsterdan, May, 1984.

/ROB 84/ Robinson, W. Maintance of digital radio using remote performance monitoring. Proceedings IEEE International Conference on Communications. Amsterdan, May, 1984.

/ROO 86/ Roos, J.D. and van den Heever, R.J. Trends in Network Management. Internation Conference on Data Communication , IFIP TC-6, Sandton Sun, 17-19,March,1986.

/STO 82/ Stolfo, S. and Vesonder, Gregg. ACE: expert system supporting analysis and management decision making. Proceedings of the Eight International Joint Conference on Artificial Intelligence, 8-12 August 1983, Karlruhe, West Germany.

GRUPO II

MEDIDAS DE ÊFICIENCIA, UM CASO DE ESTUDO

Maria Angélica Camargo
Beatriz R. Tavares Franciosi
Universidade Federal do Rio Grande do Sul
Porto Alegre - *Brasil*

INTRODUÇÃO

É através da Matemática que se traçam os possiveis caminhos da resolução de problemas científicos. Entretanto, muitas vezes, a aplicação não é imediata em um problema numérico.

A análise numérica é o ramo da Matemática que desenvolve técnicas para a resolução de problemas numéricos.

Os métodos de aproximação de solução para equações polinomiais, fornecidos pela Análise Numérica, são de vital importância para a resolução de equações de grau superior a 4 e até mesmo para algumas de ordem 3 ou 4 que não podem ter suas raízes exatamente computadas através de métodos analíticos.

Os vários métodos, determinam um número para o qual a função neste ponto é zero, ou seja, $f(\xi)= 0$ (ξ é o limite de uma seqüência de iterações).

Neste trabalho, apresentamos alguns métodos iterativos, os quais foram por nós escolhidos para amostrar um grupo maior.

O objetivo do trabalho não é o estudo individual dos métodos mas sim estima o "esforço" envolvido para completar um laço da função iterativa com a exatidão desejada. Esta medida de "esforço" será denominada INDICE DE EFICIÊNCIA DO MÉTODO e calculada segundo as funções índice de eficiência apresentadas no segmento do trabalho.

Não existe nada de inédito neste trabalho, existe sim uma tentativa de oferecer ao leitor uma análise imparcial do índice de eficiência* dos métodos apresentados.

* Índice de eficiência teórica & índice de eficiência prático

CONSIDERAÇÕES SOBRE OS MÉTODOS ANALISADOS

1) Método de Newton

Fórmula do método

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

* O cálculo do método de Newton é relativamente simples e realiza as iterações em um espaço não muito grande de tempo. Isto faz com que ele valha a pena de ser implementado.

* O método requer a avaliação da derivada e da função a cada iteração.

Existem dois métodos que podem ser usados para avaliar a derivada em um programa de computador:

1º método - incluir uma subrotina que calcule a expressão analítica para calcular a derivada.
desvantagem do método - requer uma subrotina diferente para toda nova função.

2º método - usar uma fórmula (bastante exata) aproximada, para a derivada, ou seja, por definição a derivada de f é dada por:

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

Neste caso temos que fazer algumas considerações quanto ao Δx. Como não é possível fazer x aproximadamente zero, em um programa, nós temos que reduzi-lo a um número bastante pequeno. A questão é o quanto pequeno?
A melhor solução é fazer o tamanho de Δx ser proporcional ao tamanho de x, ou seja, se x é muito grande, então Δx será muito grande. Se x é muito pequeno, então Δx é uma boa aproximação de x.
$\Delta x = \theta x$ onde θ é uma constante de proporcionalidade.

* convergência do método de Newton pode ser obtida considerando a série de Taylor.

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x - x_0)^2}{2} + \ldots$$

Considerando o valor inicial assumido para o método.

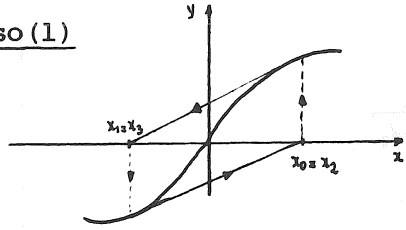
Como os métodos de Newton se aproxima da raíz ignorando os termos que sucedem à 1º derivada, temos

logo a ordem do erro é

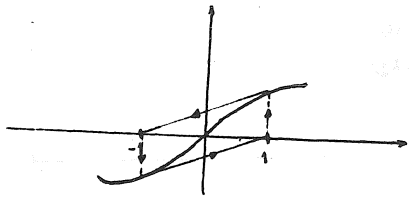
A convergência é quadrática para raízes simples e é linear para raízes múltiplas.

* problemas do método de Newton

caso(1)



caso(2)

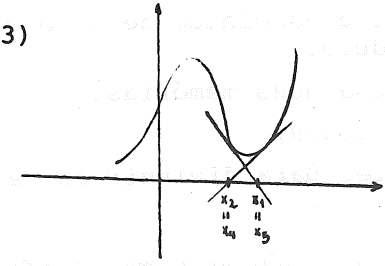


$f(x) = \frac{x}{1+|x|}$ uma raíz é zero mas para $x_0 = 1$ temos

$x_1 = -1$
$x_2 = 1$
$x_3 = -1$
⋮

$x_n = (-1)^n$ ou seja para $x_0 \geq 1$ o método diverge.

caso(3)



2) Método da Secante

Fórmula do método

$$x_{n+1} = x_n - f(x_n)\frac{(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$$

* a fórmula fornece uma maneira segura e precisa para calcular x_{n+1} se $|f(x_n)| < |f(x_{n-1})|$

  Se isto não acontecer, então troca-se x_n por x_{n-1}

* convergência super-linear (1.616)
* unipontual com memória-usa uma memória
* útil para funções com cálculo de derivada difícil
* trabalha com um intervalo inicial $[x_0, x_1]$ e a cada iteração é feito uma avaliação da função. O valor da iteração anterior é reusado.

3) Método de Müller

Fórmula do método

$$x_{n+1} = \frac{2d_2}{-d_1 \pm \sqrt{d_1^2 - 4d_0d_2}}$$

onde d_1, d_2, d_0 satisfazem

$$\begin{cases} d_0 x_0^2 + d_1 x_0 + d_2 = p(x_0) \\ d_0 x_1^2 + d_1 x_1 + d_2 = p(x_1) \\ d_0 x_2^2 + d_1 x_2 + d_2 = p(x_2) \end{cases}$$

* o sinal a frente do radical é escolhido de modo que o denominador tenha o maior módulo.
* multipontual como memória-usa duas memórias.
* convergência super-linear (1.839)
* Algoritmo proposto por Müller para diminuir os erros de arredondamento:

1. entrada {grau do polinômio, coeficientes do polinômio, 3 pontos iniciais}

2. calcula $f(x_0)$, $f(x_1)$, $f(x_2)$

3. $\lambda_2 = \dfrac{x_2 - x_1}{x_1 - x_0}$

4. Para $i = 3, 4, 5, \ldots$ até que satisfaça:

4.1. $d_i = 1 + \lambda_i$

4.2. $g_i = f_{i-2}\lambda_i^2 - f_{i-1}d_i^2 + f_i(\lambda_i - d_i)$

4.3. $\lambda_{i+1} = \dfrac{2f_i d_i}{g_i \pm \sqrt{g_i^2 - 4f_i d_i \lambda_i [f_{i-2}\lambda_i - f_{i-1}d_i + f_i]}}$

4.4. $h_{i+1} = \lambda_{i+1} h_i$

4.5. $x_{i+1} = x_i + h_i$

5. saída $\{i, x_i\}$

4) Método de Fourier

Fórmula do método

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} - \frac{f(x_n - f(x_n)/f'(x_n))}{f'(x_n)}$$

* satisfaz as condições de Fourier, isto é:

  seja f(x) uma função no intervalo $I_0 = [x_0, y_0]$ que satisfaz as condições:

  (i) $\alpha \in I_0$, onde $f(\alpha) = 0$
  (ii) $f'(x) \cdot f''(x) \neq 0 \quad \forall x \in I_0$
  (iii) $f(x_0) \cdot f(y_0) < 0$
  (iv) $f(x_0) \cdot f''(x_0) > 0$

* ordem de convergência cúbica
* o método requer a avalização da derivada e da função a cada iteração
* é necessário verificar se $x_{n+1} < y_{n+1}$ a cada iteração.
* método híbrido.

5) Método de Dandelin

Fórmula do método

$$x_{n+1} = x_n + \frac{\left(\frac{f(x_n)}{f'(x_n)} \cdot f(x_n)\right)}{f\left(x_n - \frac{f(x_n)}{f'(x_n)}\right) - f(x_n)}$$

* satisfaz as condições de Fourier.
* ordem de convergência cúbica.
* o método requer a avaliação da função e da derivada a cada iteração.
* a cada iteração é necessária a verificação $x_{n+1} < y_{n+1}$
* método híbrido

6) Método C

Fórmula do método

$$x_{n+1} = x_n - f(x_n)\,\frac{y_n - x_n}{f(y_n) - f(x_n)}$$

$$y_{n+1} = x_{n+1} - \frac{f(x_{n+1})}{f'(x_{n+1})}$$

* ordem de convergência cúbica.
* o método requer a avaliação da função e da derivada a cada iteração.
* método híbrido
* satisfaz as condições

(i) $\alpha \in X_0 = [x_0, y_0]$, onde $f(\alpha) = 0$
(ii) $f(x_0).f(y_0) < 0$

e uma das condições

(iii) $f'(x) > 0$ e $f''(x) < 0$
(iv) $f'(x) < 0$ e $f''(x) > 0$
(v) $f'(x) > 0$ e $f''(x) > 0$
(vi) $f'(x) < 0$ e $f''(x) < 0$

é necessário verificar se: $x_{n+1} < y_{n+1}$
$y_{n+1} < y_n$
$x_{n+1} < x_n$

MEDIDA DE EFICIÊNCIA - (INDICE DE EFICIÊNCIA)

O índice de eficiência nos fornece uma estimativa do "esforço" computacional envolvido para completar um laço da função iterativa. Espera-se que tal medida de eficiência possa ser obtida através da definição de uma função que meça o indice de eficiência.

Questiona-se aqui: quais os ítens que devem ser considerados para definir eficientemente a função - índice de eficiência? Que conjunto de entradas deve ser considerado quando se quer valorar o trabalho dispendido para completar um laço da iteração? como ponderar convenientemente estas entradas em uma função iterativa, de tal forma que elas nos forneça como saida, uma informação confiável? Itens como ordem de convergência, tamanho da constante assintótica de erro, custo da função e de suas derivadas por iteração é sabido que devem ser considerados, mas eles são suficientes ou necessários para formular uma função-índice de efi-

ciência-confiável?

Com a finalidade de sabermos até que ponto a teoria, ou seja, o índice de eficiência teórico revela o que acontece praticamente, foi feita a análise teórica e prática dos índices de eficiência mais encontrados na bibliografia.

MEDIDA DE EFICIÊNCIA TEÓRICA

Ostrowski(1960) $E_2 = p^{\frac{1}{e}}$ onde p: é a ordem de convergência do método
e: é o número de avaliações da função por iteração

Traub(1964) $E_1 = \frac{p}{e}$ onde p,e são obtidos análogamente à Ostrowski

Estas medidas assintóticas caracterizam um método em particular, quando o número de iteração tende ao infinito.

Quanto maior o valor do índice de eficiência mais eficiênte, assintóticamente, é o método.

As duas medidas apresentadas acima, nem sempre demonstram a realidade e deixam a desejar quanto aos ítens a ser considerados.

Feldston & Firestone(1969)

Levando em consideração que algoritmos de mesma eficiência teórica não tinham a mesma eficiência na prática Feldston & Firestone propuseram seu índice

$E_3 = p^{\frac{1}{H}}$ onde $H = e\left(1+\frac{A}{B}\right)$

p: ordem de convergência do método
e: número de avaliações da função/iteração
A: número de operações aritméticas para calcular um ciclo do algoritmo.
B: nº de operações aritméticas para calcular f, f',f'', ...

Seguindo a mesma idéia

Kung e Traub(1973) propuseram uma medida de eficiência que contraria a proposta anterior de Traub.

$$E_6 = \frac{\log_2 p}{\sum_i n_i V(f^{(i)}) + C(\emptyset)}$$

p: ordem de convergência do método

n_i: nro de avaliações de função f

$V(f^{(i)})$: nro de operações aritméticas para uma avaliação de f

$C(\emptyset)$: nro mínimo de operações aritméticas do método

i: número de derivadas da função

Peterson(1975)

$$E_4 = \frac{1}{M} \log_2 p$$

M: nro total de multiplicações da função iteração e das funções envolvidas

Ainda

$$E_5 = C \frac{\log p}{A_n + B_n}$$

An: tempo para completar a iteração sem contar f, f',f'',...

Bn: tempo para calcular f, f', f'',...

A escolha da função iterativa, com a qual nós iremos resolver nosso problema, depende da natureza do mesmo e portanto é dificil se fazer uma recomendação precisa de qual função é a mais apropriada.

Assumindo que o polinômio e suas derivadas são avaliados usando o algoritmo de Horner, o qual requer para um polinômio de grau n, n adições e n multiplicações, foi definida a tabela(1).

A construção da tabela (1) foi baseada na tabela auxiliar, na qual constam explicitamente os valores dos parâmetros de entrada tomados.

Com a finalidade de verificar o melhor método segundo os índices sugeridos (teóricamente) foi feita uma avaliação pra grau do polinômio (n) variando de 1 até 100. As conclusões a que se chegou estão no segmento do trabalho.

OBS: A listagem da avaliação está a disposição dos interessados.

TABELA (1)

Nome do método	E_1	E_2	E_3	E_4	E_5	E_6
Newton	1	1.414	$2^{\frac{2n-1}{4n+3}}$	$(2n)^{-1}$	1505/n	1/4n
Secante	1.518	1.618	$1.618^{\frac{4n}{4n+5}}$	$(2.88+2.88n)^{-1}$	104.5/(1+n)	1/2.88n
Müller	1.839	1.839	$1.839^{\frac{n}{n+14}}$	1.1099	264.6/(8+n)	0.0488
Fourier	1	1.442	$3^{\frac{3n-1}{9n+3}}$	$(1.893n+0.63)^{-1}$	477.1/(1+3n)	1/(37.94n+12.66)
Dandelin	1	1.442	$3^{\frac{6n+2}{18n+9}}$	$(1.893n+1.262)^{-1}$	477.1/(2+3n)	1/(37.94n+18.99)
C	o.75	1.316	$3^{\frac{4n-1}{16n-10}}$	$(2.524n+1.262)^{-1}$	238.6/(1+2n)	1/(50.63n+31.65)

TABELA AUXILIAR

Nome do método	P	A	B	M	An	Bn	ni	C(0)	C
Newton	2	2	4n-2	2n	1	2n-1	2	1	1000
Secante	1.618	5	4n	2n+2	2	2n	1	0	1000
Müller	1.839	28	2n	8	8	n	1	10	1000
Fourier	3	4	6n-2	3n+1	2	3n-1	2	2	1000
Dandelin	3	5	6n-2	3n+2	3	3n-1	2	2	1000
C	3	7	8n-2	3	3	4n-1	2	1	1000

A partir da tabela(1) e dos valores obtidos quando n variou de 1 até 100, concluiu-se em um primeiro momento que:

Segundo: Traub(1964) E_1 Müller é a melhor escolha.
Ostrowski(1960) E_2 Müller é a melhor escolha
Feldston & Firestone(1969) E_3 para polimios de grau:
1 a 3 Dandelin
4 a 43 Secante
47 a 100 Müller
Peterson(1975) E_4 a melhor escolha é C
... E_5 o melhor método é:
Newton para grau 1 a 5
Fourier para grau 6 a 100
Kung & Traub(1973) E_6 para polinômios de grau:
1 a 2 Newton
3 a 100 Secante

De posse destas informações resta-nos verificar até que ponto o índice de eficiência teórico e o prático se equiparam.

Para medir o tempo necessário para encontrar a raíz de um polinômio, segundo: Newton, Secante, Müller, C, Fourier, Dandelin usou-se os seguintes polinômios.

$$f_1 = (1+(1-n)^2)x-(1-nx)^2$$
$$f_2 = (1-x)^n + x^2$$
$$f_3 = (1+(1-n)^4)x-(1-nx)^4$$
$$f_4 = \frac{nx-1}{(n-1)x}$$

Usou-se uma variação para n= 1,2,3,5,15,20 e obteve-se os seguintes resultados:

Para f_1 com n=1 (melhor tempo → pior tempo)
Newton
Müller
C
Secante & Fourier
Dandelin

n=2 .....
Newton
Müller
Secante
C
Fourier
Dandelin

n=3 .....
Newton
Müller
Secante
C
Fourier
Dandelin

	n=5	Newton
		Müller
		C
		Secante
		Fourier
		Dandelin
	n=15	Müller
		Secante & C
		Newton
		Fourier
		Dandelin
	n=20	Müller
		C
		Secante
		Fourier
		Dandelin
Para f_2 com	n=1	Newton
		C
		Fourier
		Secante
		Dandelin
	n=2	Newton
		Secante, Fourier & C
		Dandelin
	n=3	Newton
		C
		Secante
		Fourier
		Dandelin
	n=5	Newton
		C
		Dedelin
		Secante
		Fourier
	n=15	Newton
		C
		Fourier
		Dandelin
		Secante
	n=20	Newton
		C
		Dandelin
		Fourier
		Secante
para f_3 com	n=1.....	Secante
		Newton
		C
		Fourier
		Dandelin
	n=2	Secante
		Newton
		C
		Fourier
		Dandelin

		n=3	Secante Newton Fourier Dandelin C
		n=5	Secante Newton Fourier Dandelin C
		n=15	Secante Newton Fourier Dandelin C
		n=20	Secante Newton Fourier Dandelin C
Com f_4	com	n=2	Secante Newton C Dandelin Fourier
		n=3	Secante Newton Dandelin Fourier C

CONCLUSÃO

* O índice de eficiência E_5, é o índice teórico que mais se aproxima da realidade;
* mesmo E_5 sendo o melhor ele nem sempre é consonante com o que acontece na prática, daí não é confiável;
* os índices de eficiência propostos, não medem tempo(custo). Algumas vezes eles coincidem com o tempo real;
* A afirmação: Quanto maior a ordem de convergência do método menor é o tempo de execução (mais rápido ele converge),
  não é confirmada na prática.
* Como os tempos são muito pequenos a escolha do método não está rigidamente ligada ao de melhor tempo mas sim ao método que dá uma garantia maior da raíz(métodos híbridos fornecem o intervalo onde se encontra a raíz e portanto o erro é menor).

COMPARANDO METRICAS DE COMPLEXIDADE DE PROGRAMAS

Ana Price
Universidade Federal do Rio Grande do Sul
Porto Alegre - *Brasil*

Introdução

Na avaliação de qualidade de um produto de software, um dos atributos mais freqüentes mencionados é a complexidade do fluxo de controle do sistema. Nos últimos anos várias tentativas tem sido feitas para derivar medidas a partir de programas-fonte que quantifiquem a noção de complexidade de programa.

O que vem a ser complexidade de programa? Encontramos na literatura várias definições para o termo, sendo que a maioria poderia ser incluída na seguinte:

"Complexidade de um programa é a medida da dificuldade de entender e trabalhar (modificar, depurar, testar) com o programa.

A medida de complexidade de um programa é importante por várias razões: (1) permite-nos estimar o esforço, tempo e custo de manutenção do programa; (2) permite-nos identificar pontos em que o programa poderá apresentar problemas e portanto re-estruturá-lo; (3) fazer comparações entre programas distintos que implementem a mesma função; e também, (4) fazer previsões do número de erros do programa.

Existem vários fatores que tem influência direta na manutenção e teste de programas, como por exemplo, o próprio tamanho do programa, a complexidade das estruturas de dados utilizadas, o fluxo de dados, o nível de embutimento das estruturas de controle, etc. Dentre as várias métricas desenvolvidas para avaliar a complexidade de programas, a complexidade de fluxo de controle e o aspecto de software mais freqüentemente medido.

Entre as métricas mais difundidas encontram-se "Software Science"" /Halstead 77/ e a Complexidade Ciclomática /McCabe 76/. A métrica de Halstead constitui-se num refinamento da medida de tamanho de programa pela contagem de linhas de código. A complexidade medida por Software Science baseia-se no número de operandos (variáveis e constantes) e operadores (aritméticos, lógicos, palavras-chave e delimitadores) que aparecem no programa. Software Science inclui também medidas quantitativas de nível de programa e de linguagem, e efeito de modularização. Prediz o tamanho de progra-

mas e estima o tempo que um programador de nível médio leva para implementar um dado algoritmo. Estudos estatísticos mostraram fortes correlações entre as predicações da teoria e medidas reais de tempo de programação e número médio de programas /Fitzsimmons 78/. Entretanto, existem críticas de que os programas usados para validar a teoria eram muito pequenos e que a base (contagem de operandos e operadores) em que Software Science se fundamenta é fraca devido a ambigüidades relativas a "o que" deve ser contado e "como" contá-lo |Shen 83|.

A métrica de McCabe define como medida de complexidade de programa o número ciclomático de um grafo que representa o fluxo de execução do programa. Segundo McCabe, a complexidade de um programa é independente de seu tamanho (número de comandos), mas depende unicamente de seu fluxo de controle. Quanto maior o número de ciclos no programa, maior a sua complexidade.

O objetivo deste trabalho é apresentar algumas métricas de complexidade de fluxo e compará-las quanto a dois aspectos relevantes da complexidade de programas: a estrutura do programa e o nível de embutimento de predicados de controle. As métricas a serem estudadas compreendem: Complexidade Ciclomática |McCabe 76|, Scope Ratio |Harrison 81|, Complexidade de Expressões Regulares |Magel 81| e Nesting Level |Piwowarski 82|.

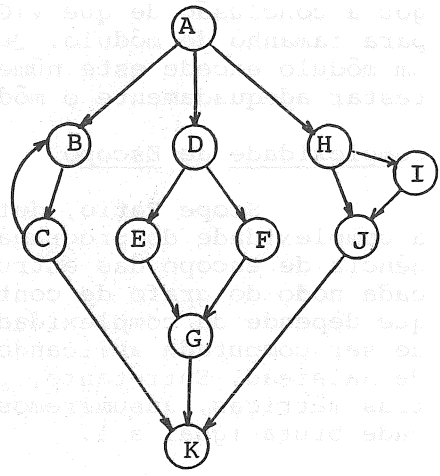
Medindo a Complexidade de Fluxo de Controle

A maioria dos trabalhos desenvolvidos sobre complexidade de software se relaciona com os efeitos do fluxo de controle sobre a complexidade de programas. Para explicar a importância do fluxo de controle com relação a complexidade, consideremos, por exemplo, um programa de 30 linhas com 15 comandos IF-THEN-ELSE. Este programa poderá conter acima de 30 mil caminhos de execução, e é obvio que tal configuração certamente será difícil de ser completamente compreendida.

As estratégias de medida de complexidade do fluxo de controle, usualmente, representam o programa como um grafo dirigido a fim de mostrar a topologia do fluxo de execução do programa. Um grafo dirigido G = (V,E) consiste de um conjunto de nodos V e um conjunto de arcos dirigidos E conectando os nodos. Num grafo de fluxo cada nodo representa um bloco de código seqüencial e os arcos correspondem ao fluxo de controle entre os vários nodos. Assumiremos que um grafo de fluxo de programa contém apenas um nodo inicial e um único nodo final, e as propriedades de que todo nodo é acessível a partir do nodo inicial, e por sua vez, o nodo final e acessível a partir de qualquer outro nodo do grafo. O grafo dirigido G1 ilustrado na Figura 2 representa o fluxo de controle do programa mostrado na Figura 1.

```
BEGIN
   CASE   exp  OF
          C1 : REPEAT  S1
               UNTIL  P1;
          C2 : BEGIN
                 IF  P2
                 THEN  S2
                 ELSE  S3;
                 S4;
               END;
          C3 : BEGIN
                 IF  P3
                 THEN S5
                 S6
               END;
    END;
  S7;
END;
```

Programa Exemplo - Figura 1



Grafo G1 - Figura 2

Complexidade Ciclomática de McCabe

McCabe afirma que a complexidade ciclomática de um grafo de programa é aplicável na determinação da dificuldade de depurar e testar o programa. O número ciclomático V(G) de um grafo dirigido G fortemente conexo é igual ao número de circuitos linearmente independentes. Num programa estruturado a complexidade ciclomática é igual ao número de condições acrescido de 1. V(G) também pode ser obtido pela fórmula:

$$V(G) = m - n + p$$

onde m = número de arcos de G, n = número de nodos de G, e

P = número de componentes individuais.

Calculando-se a complexidade ciclomática do programa exemplo (Figura 1) representado pelo grafo G1, obtemos:

$$V(G)=16 - 11 + 1 = 6$$

(Foi considerado um arco do nodo K para o nodo A a fim de tornar o grafo fortemente conexo).

Como o número ciclomático cresce com o número de caminhos de decisão e ciclos, a métrica de McCabe fornece uma medida quantitativa da dificuldade de testar um programa. Estudos experimentais indicaram relações entre a métrica de McCabe e o número de erros existentes no código fonte do programa, bem como o tempo requerido para encontrar e corrigir tais erros |Pressman 82|.

McCabe assegura que V(G) pode ser usado para fornecer uma indicação quantitativa do tamanho máximo de um módulo. A partir de dados de vários projetos reais, McCabe che

gou a conclusão de que V(G)=1Ø parece ser um limite superior para tamanho de módulo. Quando a complexidade ciclomática de um módulo excede este número, torna-se extremamente difícil testar adequadamente o módulo.

Complexidade de Escopo

Scope Ratio, definida por Harrison e Magel, deriva a complexidade do programa a partir de uma análise da abrangência de escopo das estruturas de controle. Nesta métrica, cada nodo do grafo de controle tem uma complexidade "bruta" que depende da complexidade dos comandos do nodo, e que pode ser computada aplicando-se, por exemplo, Software Science de Halstead. Entretanto, para efeitos de comparação com outras métricas, assumiremos que todos os nodos tem complexidade bruta igual a 1.

Scope Ratio considera que o grafo de um programa tem dois tipos de nodos: nodo seletor, que tem dois ou mais arcos partindo de si próprio, e nodo receptor, com apenas um arco de partida. Para obter a medida de escopo, criamos um subgrafo formado por todos os nodos que sucedem a um nodo seletor. Este subgrafo tem no mínimo um nodo, chamado "limite inferior", o qual está sobre o caminho de todos os nodos. Aquele limite inferior que precede a todos os outros limites inferiores do subgrafo é chamado "limite inferior maior" (LIM). As complexidades brutas de todos os nodos sobre os caminhos que levam ao LIM (excluindo-se a deste último) são somadas e adicionadas a complexidade bruta do nodo seletor obtendo-se a complexidade ajustada do nodo.

Este processo é repetido para cada nodo seletor do grafo de controle. A complexidade ajustada de um nodo receptor é simplesmente a sua complexidade bruta. As complexidades ajustadas de todos os nodos são então somadas, e esta soma, chamada Número de Escopo, é a complexidade do programa como um todo.

Como foi atribuída a todo nodo uma complexidade bruta de valor igual a 1, a complexidade real do programa pode ficar deturpada. Por esta razão, os autores introduziram uma medida chamada "Razão de Escopo", que é a razão entre o número de nodos do programa e o Número de Escopo. A medida que a magnitude da Razão de Escopo decresce, a complexidade do programa cresce. Por exemplo, um programa com uma razão de escopo menor do que Ø.4 é considerado razoavelmente complexo.

A tabela a seguir é o resultado da aplicação de Número e Razão de Escopo sobre o grafo do programa exemplo (Figura 2).

Seletores	Escopo	LIM	Complexidade
A	B C D E F G H I J	K	1Ø
C	B C	K	3
D	E F	G	3
H	I	J	2
			18

Receptores	Complexidade
B E F G I J K	7

Número de Escopo (NE) = 18 + 7 = 25
Razão de Escopo (RE) = 11/25 = Ø.44

Métrica de Piwowarski

Piwowarki alega que existem três aspectos segundo os quais as métricas existentes, em geral, não abragem as noções intuitivas de complexidade: (1) um programa estruturado é menos complexo que uma versão não estruturada do programa; (2) estruturas de controle embutidas são mais complexas que estruturas seqüencias de controle; e (3) um comando CASE com N alternativas é menos complexo que N-1 comandos IF embutidos |Piwowarski 82|.

"Nesting Level Complexity Measure" propõe a seguinte medida:

$$NL = V^*(G) + \sum_i P(i)$$

onde V*(G) é a complexidade ciclomática ajustada com estruturas CASE tratadas como predicados únicos. P(i) é a profundidade de embutimento do i-ésimo predicado, e tem como base o conceito de escopo de predicado definido em |Harrison 81|. A profundidade (nível) de embutimento do predicado i é o número de escopos de predicado sobrepostos ou contidos pelo i-ésimo nodo-predicado. Qualquer nodo acessível a partir do nodo-predicado, excluindo-se o "limite inferior maior" e os nodos seguintes, está dentro do escopo do nodo-predicado. A métrica de Piwowarski aplicada ao grafo da Figura 2 resultou em:

Predicados	Escopo	Complexidade	
A	C D H	3	O nodo A é o único nodo-predicado que contém outros nodos de controle.
C	-	Ø	
D	-	Ø	
H	-	Ø	
V(G) = 6	NL = 6 + 3 = 9		

Métrica de Magel

Segundo Magel, expressões regulares podem ser usadas para observar a complexidade de seqüencias possíveis de execução de um programa |Magel 81|.

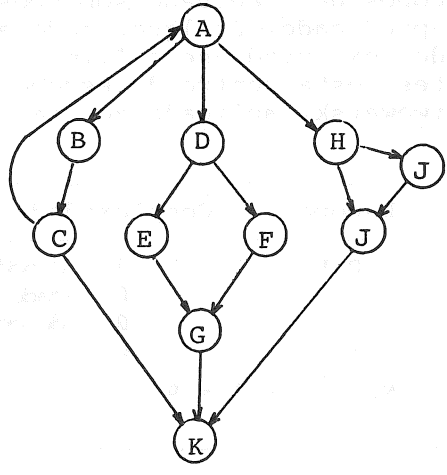
A métrica de Magel utiliza o operador de fechamento de Kleene "*" para indicar iteração e o operador de união "+" para indicar seleção. Por exemplo, a expressão regular que representa todas as seqüencias possíveis de execução para o programa exemplo da Figura 1 é a seguinte:

A (BC (BC) * + D (E+F) G + H (IJ+J)) K

A medida de complexidade de uma expressão regular é computada pela contagem do número de símbolos (operandos, operadores e parenteses) na expressão regular equivalente minimamente parentizada. Por exemplo, A(BC(BC)*+D(E+F)G+H(IJ+J))K tem complexidade 27 (14 operandos, 1 operador "*" e 4 operadores "+", e oito parenteses).

Programa Estruturado vs Não-Estruturado

Usualmente, programas estruturados (formados a partir das construções básicas Seqüencia, Seleção e Iteração) são considerados menos complexos do que programas equivalentes contendo GO-TO's. Como um dos objetivos deste trabalho é comparar as métricas apresentadas quanto a avaliação da estrutura do programa, vamos fazer uma modificação no grafo G1 (Figura 2) a fim de alterar a sua estrutura. Vamos substituir o arco CB pelo arco CA (Figura 3). Assim, ao invés do comando REPEAT-UNTIL, representado pelo ciclo entre os nodos B e C, temos agora um bloco de comandos (nodo B) seguido por um comando seletivo (C) no qual um dos arcos é um desvio incondicional (arco CA). A modificação realizada supõe a adição de um comando GO-TO ao grafo G1.



Grafo G2 - Figura 3

Aplicando-se as métricas descritas acima ao grafo modificado obtemos a seguinte tabela comparativa.

	V(G)	NE	RE	NL	ER
G1 estruturado	6	25	Ø.44	9	27
G2 contendo GO-TO	6	34	Ø.32	12	28

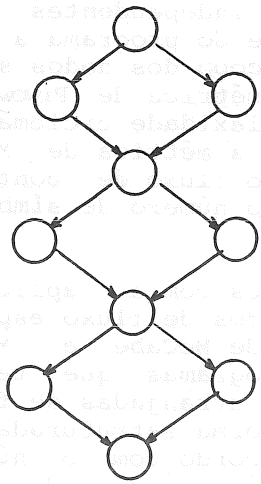
Observando-se a tabela, vemos que a complexidade ciclomática não levou em conta a modificação na estrutura do programa. Por outro lado, as métricas de escopo e de nível de embutimento acusaram um aumento considerável de complexidade com a alteração do programa. O nodo C teve sua complexidade aumentada de 3 para 11 no cálculo da Razão de Escopo pois seu escopo passou a englobar todos os nodos do grafo exceto o nodo K. No caso da métrica de Nível de Embutimento, a complexidade do nodo C cresce de Ø para 3. Por último a Métrica de Magel resultou na expressão regular

A (BCA) * (BC+D (E+F) G+H (IJ+J)) K

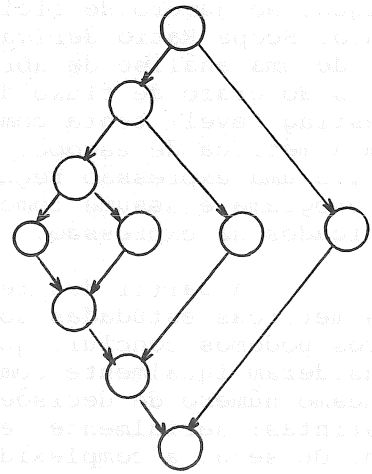
ligeiramente mais complexa que a anterior.

Embutimento de Estruturas de Controle

O embutimento de estruturas de controle é considerado por vários autores como fator incremental na complexidade de programas: quanto maior o embutimento de precidados, maior é o grau de complexidade do programa |Harrison 81, Piwowarski 82|.



Grafo G3



Grafo G4

Figura 4

Os grafos de fluxo G3 e G4 poderiam representar dois programas diferentes com blocos de comandos sequenciais identicos porém com as estruturas de controle arranjadas diferentemente. Estes arranjos podem afetar consideravelmente a complexidade dos programas. No programa representado pelo grafo G3 as decisões são seriais enquanto que no outro temos dois níveis de embutimento de predicados. A tabela abaixo nos mostra as complexidades calculadas para os dois grafos:

	V(G)	NE	RE	NL	RE
G3	4	16	Ø.62	4	19
G4	4	25	Ø.40	7	19

Da tabela podemos inferir que ambas as métricas de Complexidade Ciclomática e Expressões Regulares consideram igualmente complexos os dois programas. Isto é, consideram que o embutimento de estruturas de controle não altera a complexidade do programa. As métricas de escopo e nível de embutimento acusaram um aumento de complexidade considerável.

Conclusões

As métricas de complexidade de fluxo de controle de programas trabalham, geralmente, sobre grafos dirigidos os quais representam fluxos de controle de programas.

Foram descritas quatro métricas de complexidade: Complexidade Ciclomática de McCabe, Scope Ratio de Harrison e Magel, Nesting Level de Piwowarski e a Métrica de Magel sobre Expressões Regulares. McCabe define como medida de complexidade, o número ciclomático do grafo do programa, o qual é igual ao número de ciclos linearmente independentes no grafo. Scope Ratio deriva a complexidade do programa a partir de uma análise de abrangência de escopo dos nodos seletores do grafo de fluxo de controle. A métrica de Piwowarski (Nesting Level) tenta combinar a complexidade ciclomática com a métrica de escopo. E finalmente, a métrica de Magel deriva uma expressão regular a partir do fluxo de controle do programa e assume como complexidade o número de símbolos indicados na expressão.

A partir dos resultados obtidos com a aplicação das métricas estudadas sobre quatro grafos de fluxo específicos podemos concluir que as métricas de McCabe e Magel consideram igualmente complexos dois programas que tenham o mesmo número de decisões. porém estas arranjadas de formas distintas: serialmente, embutidas, de forma estruturada ou não. Ou seja, a complexidade varia de acordo com o número de decisões não importanto como estas são posicionadas no fluxo de controle do programa. Por outro lado, as métricas de escopo e nível de embutimento acusarem um acréscimo considerável na complexidade dos programas quando da inserção

de GO-TO e embutimento de estruturas de controle.

As métricas de fluxo de controle, tais como quaisquer outras classes de medidas de complexidade, são falhas porque não levam em conta qualquer outro fator que não seja a complexidade do fluxo de execução do programa. Entretanto, elas fazem um bom trabalho ao diferenciar dois programas que do contrário seriam equivalentes em outras características, tal como o tamanho.

Agradecimentos

A autora agradece os incentivos recebidos da FINEP e CNPq.

Referências Bibliográficas

|Fitzsimmons 78|
A. Fitzsimmons e T. Love, "A Review and Evaluation of Software Science", ACM Computing Surveys, Março 1978, pag. 3-18.

|Halstead 77|
M. Halstead, "Elements of Software Science", Elsevier North-Holland, Inc., New York, 1977.

|Harrison 81|
W. Harrison e K. Magel, "A Complexity Measure based on Nesting Level", ACM SIGPLAN Notices, Vol. 6, Nº.3 , Março 1981, pag. 63-74.

|Harrison 82|
W. Harrison, K. Magel, R. Kluczny e A. DeKock, "Applying Software Complexity Metrics to Program Maintenance, Computer, Setembro 1982, pag. 65-79.

|Magel 81|
K. Magel, "Regular Expressions in a Program Complexity Metric", ACM SIGPLAN Notices, Julho 1981, pag. 61-65.

|McCabe 76|
T. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Dezembro 1976, pag. 308-320.

|Piwowarski 82|
P. Piwowarski, "A Nesting Level Complexity Measure", ACM SIGPLAN Notices, Setembro 1982, pag 44-50.

|Pressman 82|
R. Pressman, "Software Engineering: A Practitioner's Approach", McGraw-Hill, Inc., 1982.

|Shen 83|
V. Shen, D. Samuel e H. Dunsmore, "Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support", Março 1983, pag. 155-165.

ALGORITMOS APROXIMATIVOS: UMA ALTERNATIVA PARA PROBLEMAS NP - COMPLETOS

Laira Vieira Toscani - Jayme Luiz Szwarcfiter
Universidade Federal do Rio Grande do Sul
Universidade Federal do Rio de Janeiro
Brasil

1. INTRODUÇÃO

A análise da complexidade de um algoritmo é segundo Aho e Ullman /AHO 74/ o coração da Ciência da Computação.

Várias medidas de complexidade de um algoritmo surgiram historicamente, mas sem dúvida uma das mais importantes é a medida de tempo /COO 83/. Esse trabalho tratará da complexidade de tempo de algoritmo, usando o termo complexidade, simplesmente.

A complexidade de um algoritmo em muitos casos não depende só do tamanho da entrada, mas de certas propriedades da entrada, como por exemplo no problema de ordenar uma lista de números. Ordenar uma lista em que quase todos seus elementos estão ordenados, não requer o mesmo esforço necessário para ordenar uma lista de mesmo tamanho, mas com os elementos em grande desordem. Neste caso convém estudar a complexidade do caso médio, que é a média ponderada da complexidade de cada entrada possível e a probabilidade de sua ocorrência. A complexidade do caso médio pode ser mais significativa que a complexidade no pior caso, como no estudo de algoritmos de procura heurística, em Inteligência Artificial. Para esses algoritmos o pior caso tende a ser pessimista, todo algoritmo tem um caso em que funciona de maneira muito enificiente. Além disso é difícil definir limites precisos do pior caso, sendo uma análise probabilística mais natural /HUY 80/. Seguidamente, entretanto, encontrar a distribuição probabilística, para as instâncias, que melhor se adapta ao caso, não é fácil, pois a distribuição pode mudar de maneira imprevisível com o tempo, dificultando a análise da complexidade média. Além disso essa análise não nos diz coisa alguma sobre o comportamento de um algoritmo para uma instância particular. Neste trabalho será analisado somente a complexidade no pior caso, que por simplicidade será chamada só complexidade.

A identificação de P(conjunto dos problemas resolvíveis deterministicamente por algoritmos de complexidade polinomial) como a classe dos problemas tratáveis tem sido, em geral aceita /COO 83/, apesar de um algoritmo $O(n^{1000})$ ser um algoritmo bem ruim, um problema que não possui algoritmo polinomial certamente é intratável.

Outras classes importantes de problema são: NP, conjunto dos problemas resolvíveis não deterministicamente por algoritmo de complexidade polinomial e NP-completo, conjunto de problemas NP com a propriedade adicional de que a existência de algoritmo polinomial e determinístico, que o resolva, implica em P = NP ($P \subseteq NP$ trivialmente).

O fato de existirem centenas de problemas na classe NP-completa fortifica a conjectura de que P ≠ NP.

A pertinência de um problema à classe NP - completa é segundo Cook /COO 83/, para efeito prático o limite inferior de complexidade do problema, o qual pode ser interpretado como intratável.

Existem muitos problemas NP - completos na área de Otimização, Teoria dos Grafos e de Pesquisa Operacional. Os algoritmos conhecidos que os resolvem são portanto de complexidade não polinomial, o que torna-os impraticáveis para instâncias grandes. Felizmente muitas aplicações que requerem soluções para esses problemas não exigem uma solução exata. Baseados nesse fato osprojetistas de algoritmos desenvolveram novos métodos de solução de problemas, que encontram as soluções aproximadas, esses métodos são chamados algoritmos aproximativos ou heurísticos.

As duas principais classes de algoritmos aproximativos são: uma que garante sempre uma solução próxima da solução procurada e outra que produz uma solução ótima ou quase ótima, quase sempre /WEI 77/. Essa segunda é a classe dos algoritmos probabilísticos de grande utilidade em Criptoanálise. Este trabalho tratará somente da primeira classe de algoritmos, que serão chamados pelo nome genérico de algoritmos aproximativos.

2. ALGUMAS DEFINIÇÕES

Todas definições apresentados nessa secção são baseados nos trabalhos de /GAR 79/ e /HOR 78/.

Um problema de otimização combinatorial pode ser um problema de minimização ou maximização e é caracterizado por uma terna constituída por um conjunto de instâncias, uma função candidata e uma função valor de solução.

Assim, se π é um problema de otimização, então $\pi = (D_\pi, S_\pi, m_\pi)$, onde D_π é o conjunto de instâncias de π; S_π é a função que associa a cada instância $I \in D_\pi$ um conjunto finito $S_\pi(I)$ de candidatos a solução para I; e m_π função que atribui a cada instância $I \in D_\pi$ e cada candidata a solução $\sigma \in S_\pi(I)$ um número racional positivo $m_\pi(I,\sigma)$, chamado valor solução para σ.

Seja π um problema de otimização (minimização/maximização), uma solução ótima para uma instância $I \in D_\pi$ é uma candidata a solução $\sigma^* \in S_\pi(I)$ de melhor valor, i.é tal que para todo $\sigma \in S_\pi(I)$, $m_\pi(I,\sigma^*) \leq m_\pi(I,\sigma)$ $(m_\pi(I,\sigma^*) \geq m_\pi(I,\sigma))$. E chama-se $OPT(I)$ a $m_\pi(I,\sigma^*)$.

Um algoritmo A é dito um algoritmo aproximativo para um problema $\pi = (D_\pi, S_\pi, m_\pi)$ sss para qualquer ins-

tância $I \in D_\pi$, A encontra $\sigma \in S_\pi(I)$ uma candidata a solução. E se diz que $A(I) = m_\pi(I,\sigma)$.

Um algoritmo A tal que para toda instância $I \in D_\pi$, $A(I) = OPT(I)$ é dito algoritmo de otimização para π.

3. MEDIDAS DE QUALIDADE DE ALGORITMOS APROXIMATIVOS

Os algoritmos aproximativos produzem uma aproximação da solução exata, mas então é preciso quantificar esta proximidade, para medir a qualidade do algoritmo.

Sejam π um problema de minimização (ou maximização), I uma instância de $\pi(I \in D_\pi)$ e A um algoritmo aproximativo para π, a razão $R_A(I)$, qualidade de A para I, é definida por

$$R_A(I) = \frac{A(I)}{OPT(I)} \quad (\text{ou } R_A(I) = \frac{OPT(I)}{A(I)}),$$

a qualidade absoluta de A, R_A é dada por:

$$R_A = \inf\{r \geq 1 \ / \ R_A(I) \leq r \text{ para toda instância } I \in D_\pi\}$$

e de qualidade assintótica R_A^∞ é dada por

$$R_A^\infty = \inf\{r \geq 1 \ / \text{ para algum } N \in Z^+, R_A(I) \leq r \text{ para todo } I \in D_\pi \text{ satisfazendo } OPT(I) \geq N\}$$

Muitas aplicações exigem uma qualidade mínima para aceitação de um algoritmo. Esta exigência é posta como requerimento de exatidão $\varepsilon > 0$, apartir dos quais são definidos esquemas aproximativos. É usado o termo esquema, porque para cada ε é definido um algoritmo A_ε.

Dado um problema π, um esquema aproximativo é um algoritmo A tal que dado um requerimento de exatidão ε, deriva um algoritmo A_ε tal que para uma instância $I \in D_\pi$, $R_{A_\varepsilon}(I) \leq 1 + \varepsilon$.

4. EXEMPLO

Para ilustrar é apresentado aqui um algoritmo aproximativo para o Problema do Mínimo Equivalente em Grafos (MEQ), esse algoritmo é devido a J. L. Szwarcfiter /SZW 85/.

Problema MEQ: "dado um digrafo D, encontrar o menor subgrafo gerador de D que preserve sua alcançabilidade".

Este problema é NP-difícil. O algoritmo aproximativo de J. L. Szwarcfiter tem $R_A \leq 2$.

4.1 - Algoritmo

O algoritmo foi desenvolvido para dígrafos fortemente conexos, mas uma modificação é sugerida para adequá-lo a dígrafos não fortemente conexos.

Algoritmo

entrada: D = (V,E)

1. $C \leftarrow \emptyset$; $i \leftarrow 1$;
2. Escolha arbitrariamente um ciclo C_1 em D e faça
3. $D_1 \leftarrow C_1$; $C \leftarrow C \cup \{\text{nodos de } C_1\}$;
4. Enquanto $C \neq V$ faça
5. Apartir de v encontre w em D e P(v,w) t.q. v, w $\in$ C e
6. exceto pelos extremos v e w, P(v,w) não contém
7. nodos de C;
8. $C_{i+1} \leftarrow$ P(v,w) + um caminho de w a v de D_i.
9. $D_{i+1} \leftarrow D_i + C_{i+1}$;
10. $C \leftarrow C \cup \{\text{nodos de } C_{i+1}\}$;
11. $i \leftarrow i + 1$;
12. fim-enquanto
13. fim-com-saída: D_i

Se o digrafo não é fortemente conexo, aplique o algoritmo para cada componente fortemente conexo e depois inclua as arestas (s_i, s_j) de D, sendo s_i e s_j nodos de componentes fortemente conexos diferentes.

4.2 Complexidade do algoritmo

A linha 2 tem complexidade $O(\max\{|V|,|E|\})$. Na linha 5, apartir de v é construido um caminho P(v,w), até atingir um nodo w e D. Neste caminho cada aresta de D é considerada no máximo uma vez. A complexidade total do algoritmo é então $O(\max\{|V|,|E|\})$.

4.3 Qualidade do algoritmo

Quanto a qualidade do algoritmo se pode considerar dois casos:

Caso 1: D fortemente conexo

A saída do algoritmo, D_i, é o digrafo constituído dos ciclos $C_1, C_2, \ldots, C_i$. O primeiro ciclo C_1 cobre

$n_1 > 1$ nodos e tem n_1 arestas. Cada ciclo subsequente C_j $1 < j \leq i$, cobre $n_j > 1$ novos nodos (não cobertos por C_1, C_2, ..., C_{i-j}) e tem precisamente $n_j + 1$ arestas não pertencentes a ciclo algum entre C_1, C_2, ..., C_{j-1}.

$$\sum_{j=1}^{i} n_j = |V|$$

E o número de arestas de D_i é $(\sum_{j=1}^{i} n_j+1)-1 =$

$$= i+\sum_{j=1}^{i} n_j-1$$

$$= i + |V| - 1$$

$$\leq 2\,|V| - 1$$

Então a solução encontrada pelo algoritmo aproximativo é $S \leq 2\,V - 1$, enquanto a solução ótima é $S^* \geq |V|$

Logo $R_A = \frac{S}{S^*} \leq \frac{2|V|-1}{|V|} < 2$

Caso 2: D não é fortemente conexo

Se o digrafo não é fortemente conexo as arestas que conectam os diferentes componentes ocorrem na solução aproximada, na mesma freqüência que aparecem na solução exata, portanto a relação $R_A < 2$ se mantêm.

5. RESULTADOS CORRELATOS

Algoritmos aproximativos resolveram satisfatoriamente muitos problemas, mas para outros problemas não foi possível obter-se um bom algoritmo aproximativo, outros ainda possuem algoritmos pseudopolinomiais. Neste secção estas questões são consideradas.

5.1 Problemas que não admitem algoritmo aproximativo polinomial

O problema do Caixeiro Viajante é um dos mais conhecidos problemas NP-completos e pode ser definido assim:

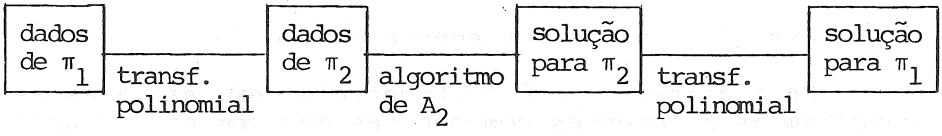
Problema C V: Seja um conjunto finito $C = \{c_1, c_2, \ldots c_n\}$ de cidades e $d(c_i, c_j) \in Z^+$ a distância entre as cidades c_i e $c_j \in C$. A questão é achar o menor percurso para o Caixeiro Viajante, que saindo de uma das cidades visite todas as outras cidades exatamente uma vez e retorne a cidade de origem.

Este problema, interpretado na Teoria dos Grafos consiste em achar um circuito hamiltoniano de custo mínimo, para o grafo G = (V,E), onde V=C conjunto de cidades e E, o conjunto de arestas, é definido como o conjunto de todos pares (c_i, c_j) tal que $d(c_i, c_j) < \infty$, d dá o custo de cada aresta e o custo de um caminho em G é a soma dos custos de cada aresta do caminho.

Através da Programação Dinâmica é possível projetar um algoritmo para o problema CV com complexidade $0(n^2\ 2^n)$ /TOS 86/ que é bem melhor que o algoritmo trivial que o enumera os n! permutações dos n nodos do grafo. Mas seria interessante se obter um algoritmo aproximativo de complexidade polinomial. Infelizmente a existência de tal algoritmo implica em P=NP /GAR 79/.

5.2 Transformação Polinomial

Dados dois problemas NP-completos π_1 e π_2, π_1 pode ser transformado polinomialmente em π_2, segundo o diagrama abaixo e o algoritmo A_2 que resolve π_2 pode ser usado para resolver π_1.



Assim, se π_2 tem algoritmo polinomial que o resolva, π_1 também tem.

Logo, é de se esperar que se π_2 tem um algoritmo aproximativo A_2' de tempo polinomial, o mesmo método gere um algoritmo aproximativo polinomial, A_1', para π_1. Infelizmente isto não acontece.

Sejam os dois seguintes problemas de Teoria dos Grafos:

Problema do Máximo Conjunto Independente (MCI): Dado um grafo G = (V,E), $V' \leq V$ é um conjunto independente se $\forall u, v \in V$ $(u,v) \notin E$. Um conjunto máximo independente é um conjunto independente V_{max} tal que para todo conjunto independente V', $V' \not\subseteq V_{max}$.

Problema da Cobertura de Vértices Mínima (CVM): Dado um grafo G = (V,E), $V' \subseteq V$ é uma cobertura de vértices de G se $\forall$ $(u,v) \in E$, $u \in V'$ ou $v \in V'$. Cobertura de vértices mínima é uma cobertura de vértices V_{min} tal que para toda cobertura de vértice V', $V' \not\subseteq V_{min}$.

Estes dois problemas: MCI e CVM são NP-completos e estão intimamente ligados da seguinte forma: se G=(V,E) é um grafo e $V' \subseteq V$ é máximo conjunto independente de G, V - V' é uma cobertura de vértices mínima para G. Entretanto o problema MCI não possui algoritmo aproximativo com $R_A < \infty$ enquanto que para o problema CVM tem algoritmo aproximativo com $R_A \leq 2$ /GAR 79/.

Para exemplificar como a transformação que preserva a otimalidade de uma solução não necessariamente preserva a qualidade de uma solução aproximada /GAR 79/ apresenta o seguinte caso: suponha que se tem um grafo G=(V,E), com 1000 vértices ($|V|$ =1000), a cobertura de vértices mínima para G tem 490 vértices e o algoritmo aproximativo A, para o problema tem $R_A \leq 2$ portanto a solução encontrada pelo algoritmo, V' é tal que $|V'| \leq 980$. Usando a transformação entre os problemas apresentada antes (máximo conjunto independente = V - V'), chega-se a seguinte razão:

$$R_A(I) = \frac{1000-490}{1000-980} = \frac{510}{20} = 25,5$$

5.3 Algoritmos pseudopolinomiais

A complexidade de um algoritmo é calculada em função do tamanho da entrada, mas então depende da codificação da entrada. Um número k codificado em binário é representado por $\lfloor \log_2 k \rfloor + 1$ dígitos. Esta codificação é tradicionalmente aceita e uma codificação em uma base maior que dois é também aceita porque não altera a grandeza da complexidade do algoritmo, i.é se no primeiro caso, um algoritmo é de complexidade polinomial ou exponencial, continua sendo polinomial ou exponencial se a base de codificação for alterada para uma base maior. Mas se for utilizada a representação unária. O resultado não é o mesmo. Este enfoque propicia a definição de outros problemas.

Um algoritmo A é dito de complexidade pseudopolinomial, quando sua complexidade é polinomial para o tamanho da entrada, quando a entrada é codificada em unário.

Um problema NP-completo que admite algoritmo pseudopolinomial é dito "NP-completo fraco". Existem também problemas cuja possível existência de algoritmo pseudopolinomial implica na igualdade P=NP. Esses problemas são chamados "NP-completos forte". O problema do Caixeiro Viajante é NP-completo forte /GAR 79/.

Um exemplo de problema pseudopolinomial é o problema da partição /GAR 79/.

Existem problemas cujos valores dos dados de entrada crescem polinomialmente com a quantidade de dados, nesses casos um algoritmo pseudopolinomail é na verdade um

algoritmo polinomial e se o problema é NP-completo é NP--completo forte. Um exemplo, é o problema do circuito Hamiltoniano: a entrada é um grafo G(V,E), o maior valor de um dado, então, não necessita exceder de n = |V| ou m = |E| e a quantidade de dados é max {m,n}, mesmo se codificada em unário /SZW 84/.

Um resultado interessante foi obtido por Ibarra e Kim, que converteram um algoritmo pseudopolinomial para o problema da Mochila num algoritmo aproximativo com uma perda limitada de exatidão /GAR 79/.

A idéia é utilizar um método de desenvolvimento de algoritmos aproximativos que Horowitz em /HOR 78/ chama de "rouding", que consiste em dada uma instância de um problema de otimização encontrar outra instância tal que a solução ótima do segundo interpretado na instância original é uma boa aproximação da solução da instância original. E a solução ótima da instância modificada é obtida em tempo polinomial. Esse método pode ser aplicado a qualquer problema de otimização do tipo: Calcular

$$\max_{x} \sum_{i=1}^{n} p_i x_i \text{ restrito à } \sum_{i=1}^{n} a_{ij} x_j \leq t \quad 1 \leq j \leq n, x_i = 0$$

ou 1, $1 \leq i \leq n$, p_i, $a_{ij} \geq 0$.

6. CONCLUSÕES

Muitos problemas importantes, de grande aplicabilidade são NP-completos, portanto os algoritmos conhecidos que os resolvem são ineficientes, assim duas alternativas se apresentam:

- Evitar a procura exaustiva, fazendo uma escolha inteligente, evitando soluções parciais que certamente não levam a uma solução total. Nesse sentido é muito útil a técnica de desenvolvimento de algoritmo Programação Dinâmica /TOS 86/ e /HOR 78/.

- Para problemas de otimização, ao invés de procurar a solução ótima, procurar uma boa solução em um tempo razoável, i. é procurar um algoritmo aproximativo. Duas técnicas são usadas para projeto de algoritmos aproximativos. Uma consiste em construir iterativamente conjuntos de soluções parciais, dividir o domínio das soluções parciais em intervalos, escolher uma solução em cada intervalo, até alcançar uma solução total na vizinhança (tão próxima quanto se queira) da solução ótima. Outra é modificar a instância original de maneira a, em tempo polinomial encontrar a solução ótima da instância modificada, que deve ser uma boa aproximação (tão boa quanto se queira) da solução ótima da instância original /HOR 78/.

Através de algoritmos aproximativos muitos problemas, como o exemplo da secção 5, estão satisfatoriamente resolvidos.

Infelizmente existem também os problemas que não admitem bons algoritmos aproximativos em tempo polinomial, a não ser se P = NP, como o problema do Caixeiro Viajante apresentado na secção 5.1. Assim o problema de achar um algoritmo aproximativo polinomial, de qualidade assintótica limitada para o CV é então também um problema intratável. Para o problema de coloração de grafos não se conhece algoritmo aproximativo com qualidade absoluta limitada ($R_A < \infty$).

Outros problemas, ainda, estão numa classe meio nebulosa, como o de decidir se duas expressões regulares são equivalentes e o de decidir se uma dada palavra é gerada por uma certa gramática sensível do contexto. Esses problemas são NP-difíceis, i.é são reduzíveis polinomialmente a problemas NP-completos. Como problemas NP-difíceis tem a propriedade de não serem resolvidos em tempo polinomial a não ser se P=NP, mas não se sabe se são eles, problemas NP ou não /WEI 77/. Ou problemas como: decidir se dois grafos são isomorfos, ou dado um inteiro, saber se ele é primo. Esses são problemas NP, mas não foi possível reduzí-los polinomialmente a um problema NP-completo, nem se conhece algoritmos polinomiais que os resolva.

Esforços estão sendo dispendidos no sentido de estender os resultados positivos obtidos na área, como transformar um algoritmo pseudopolinomial em um algoritmo aproximativo polinomial, com perda limitada de qualidade; aplicar técnicas de otimização para melhorar algoritmos aproximativos e transformar algoritmos aproximativos, para aplicá-los a outros problemas. Nem sempre, entretanto, os resultados são os esperados. A existência ou não de algoritmos aproximativos de qualidade assintótica limitada parece não respeitar o fato de muitos problemas estarem intimamente relacionados por uma transformação polinomial, não sendo possível usar a mesma transformação para transformar um algoritmo aproximativo para um problema em um algoritmo aproximativo para o outro, sem perda considerável de qualidade, como no exemplo da secção 5.2.

No fracasso de uma tentativa de alcançar um resultado positivo (como encontrar um bom algoritmo aproximativo, ou transformar um algoritmo pseudopolinomial em um bom algoritmo aproximativo, ou ...) é tentado provar que este resultado só poderá ser atingido se P=NP. Alcançada essa prova, o caso é considerado fechado.

Na área de Escalonamento de Tarefas muitos são os problemas NP-completos /GAR 79/, /HOR 78/, /PIN 83/. Algoritmos aproximativos podem ser claramente a melhor solução, como por exemplo para o problema de encontrar o escalonamento ótimo de tarefas com penalidades, no caso em que

o custo em tempo de computador necessário para encontrar a solução ótima excede a maior penalidade.

Para outros problemas os algoritmos aproximativos não tem uma vantagem tão clara, mas para instâncias relativamente grandes são a única solução viável.

BIBLIOGRAFIA

/AHO 74/ AHO, A.V.; HOPCROFT, J.E. & ULLMAN, J.D. The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, 1974.

/COO 83/ COOK, S.A. An overview of Computational Complexity CACM 26,6, 1983. pp401-7.

/GAR 79/ GAREY, M.R.; JOHNSON, D.S. Computers and Intractability. A guide to the Theory of NP-Completeness, W.H. Freeman, San Francisco, 1979.

/HOR 78/ HOROWITZ, E., SAHNI, S. Fundamentals of Computer Algorithms, Computer Science, Press Rockville, 1978.

/HUY 80/ HUYN, N.; DECHTER, R. & PEARL, J. Probabilistic Analysis of the Complexity of A* Artificial Intelligence. 15(1980). pp241-54.

/PIN 83/ PINHO, A de A. Algoritmos Polinomiais Exatos ou Aproximativos para certos Problemas em Scheduling Tese de Doutorado. UFRJ, Rio de Janeiro, 1983.

/SZW 84/ SZWARCFITER, J.L. Grafos e Algoritmos Computacionais. Editora Campos, Rio de Janeiro, 1984.

/SZW 86/ SZWARCFITER, J.L. On Digraphs with a Rooted Tree Structure. Network vol. 15 (1985) 49-57.

/TOS 86/ TOSCANI, L.V. & VELOSO, Paulo, A.S. Especificação Formal e Análise da Complexidade da Programação Dinâmica. Porto Alegre, CPGCC/UFRGS, 1986.

/WEI 77/ WEIDE, B. A Survey of Analysis Techniques for Discrete Algorithms. Computing Surveys, 9,4. 1977. pp.291-313.

GRUPO III

HERMES: UN SISTEMA DE CORREO ELECTRONICO INTERUNIVERSITARIO

F. Aurtenechea, M. Hidalgo
Pontificia Universidad Católica de Chile
Santiago - *Chile*

1. INTRODUCCION

En la actualidad, el auge en el uso de la computación y la incorporación de redes de comunicación de datos, constituye una combinación muy útil para proveer, a distintas instituciones computacionales, mecanismos para compartir e intercambiar información.

Una aplicación fundamental para integrar usuarios de diversos computadores es el sistema de correo electrónico o correo basado en computadores [1,3,9,11,12,13]. Este trabajo describe el diseño de HERMES, un sistema de correo electrónico, cuya principal característica es su simplicidad y carencia de administración centralizada.

HERMES fue desarrollado en el Departamento de Computación de la Universidad Católica de Chile. Este primer diseño pretende dar los primeros pasos hacia una versión más completa y estándar para operar en el ámbito interuniversitario. Un servicio de intercambio de mensajes es fundamental para facilitar tanto el intercambio de información entre académicos e investigadores como el desarrollo de proyectos interuniversitarios.

En esta primera versión se han sacrificado ciertos aspectos de funcionalidad por simplicidad y transportabilidad del sistema. La simplicidad de su diseño hacen de HERMES un sistema que puede ser fácilmente implementado en diversas máquinas. La heterogeneidad de equipamiento computacional representa un escenario típico en las universidades. Esta primera versión de HERMES integra en funcionamiento a equipos con sistema operativo UNIX y VAX/VMS. Este tipo de equipos es muy utilizado en el ambiente universitario, lo que representa para HERMES una cobertura interesante de uso.

En un servicio de correo electrónico, la unidad de operación es la carta electrónica o mensaje de correo. HERMES provee un editor básico mediante el cual el usuario puede crear dichos mensajes para su posterior despacho. Además, integra la transferencia de archivos, siendo el recipiente de destino de éstos el mismo que él de los mensajes.

HERMES provée un servicio de despacho de mensajes certificados. Esto significa que el usuario puede solicitar la notificación del éxito o fracaso del despacho de ciertos mensajes. Si el mensaje llega correctamente a destino, el usuario recibe un reconocimiento positivo (mensaje tipo ACK) desde el nodo destino. En caso contrario, el usuario recibe un reconocimiento negativo (mensaje tipo NACK), ya sea desde el nodo destino (p.ej. si el usuario destino no existe) o desde algún nodo de enrutamiento (p.ej. si en la tabla de enrutamiento de éste, el nodo destino no existe).

Un nodo en HERMES es identificado por el par <Institución, Computador>, lo que representa el conjunto de instituciones de educación superior y sus recursos computacionales. Cada nodo puede mantener varias conexiones con nodos adyacentes, y además es responsable de mantener tablas para identificar los distintos nodos

destinatarios. Además, cualquier nodo puede operar en la modalidad de "store and forward" [9,11], almacenando mensajes en tránsito para su posterior despacho a otro destino. En este caso el nodo pasa a ser un nodo de enrutamiento. El enrutamiento es estático, mantiendo cada nodo una tabla de enrutamiento cuyas filas representan los destinos posibles y las columnas las diferentes líneas de salida, en orden de prioridad.

La transferencia de mensajes entre nodos se realiza en forma automática, a intervalos de tiempo establecidos para cada nodo en particular. Esta transferencia se efectúa en la modalidad maestro-esclavo. El nodo maestro es el que inicia la sesión de comunicación con el otro nodo (el nodo esclavo), activando en ambos nodos los procesos que realizan la transmisión y recepción de mensajes. Este esquema se repite por cada nodo adyacente al nodo maestro. Cualquier nodo HERMES puede ser un nodo maestro.

La comunicación es asíncrona a través interfaces RS232-C, ya sea mediante líneas locales ("null modems") o líneas telefónicas discadas o dedicadas. En este último caso, se proveen mecanismos para realizar discado y respuesta automática entre nodos remotos.

El modelo de comunicaciones usado contempla básicamente tres niveles. El nivel superior, denominado nivel de aplicación, provée los mecanismos básicos de interacción con el usuario para la creación, consulta y recepción de mensajes. El nivel medio o nivel de transporte ejecuta las solicitudes del nivel de aplicación organizando la correspondencia para su posterior despacho. Finalmente, el nivel bajo o nivel de comunicación es el responsable de interactuar con el medio y los protocolos de comunicación para permitir la transferencia de mensajes entre una máquina y otra. Este modelo representa una versión simplificada del modelo ISO [7] de redes de comunicación de datos.

En la actualidad, parte de este modelo está implementado usando protocolos y sistemas de comunicación existentes. Al respecto cabe señalar que el protocolo de comunicaciones corresponde a una versión modificada del protocolo KERMIT [4,5]. Por otra parte, el despacho final de correspondencia al usuario se realiza mediante un proceso (Cartero) que interactúa con el correo electrónico local (Mail) de cada computador en particular.

2. COMPONENTES BASICOS DE HERMES

A continuación se describe la arquitectura general y las componentes lógicas involucradas en el diseño de HERMES.

2.1 Arquitectura General

Cada nodo de HERMES es una entidad autónoma, que maneja un servicio simple de enrutamiento de mensajes entre los demás nodos. Las funciones de cada nodo son fundamentalmente, edición y almacenamiento de correspondencia, certificación de mensajes, despacho local y redirección remota de correspondencia, y transmisión y *scheduling* automático de mensajes entre nodos adyacentes.

En la figura 2.1 se muestra las principales componentes de la arquitectura HERMES. En base a esta figura, a continuación se describe la operación general de un nodo HERMES, en función del flujo de información que se maneja y de los procesos involucrados. Los procesos y componentes fundamentales de esta arquitectura se describen más adelante.

(1) El usuario prepara un mensaje con ayuda del editor de HERMES.
(1)' El usuario puede reenviar o cancelar un mensaje certificado que recibió un NACK.

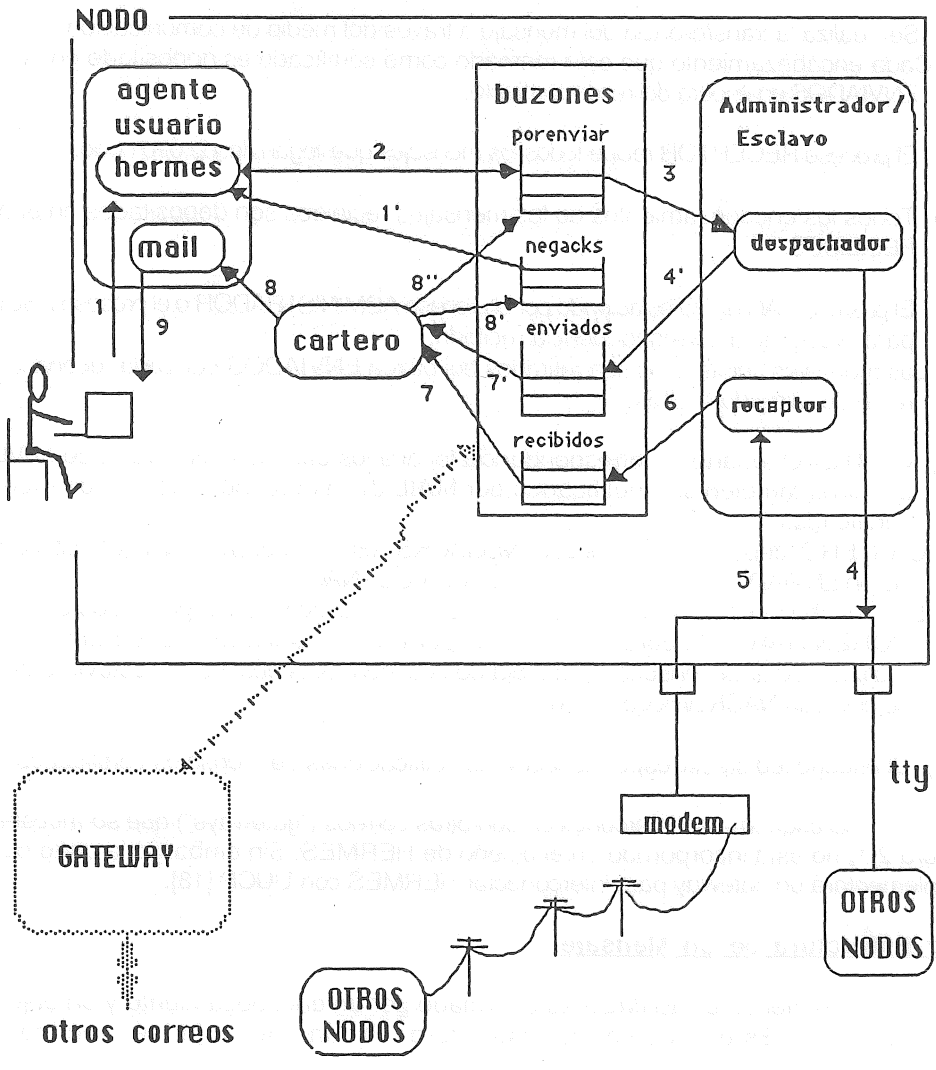


Figura 2.1 Operación General de un Nodo HERMES

(2) El encabezamiento del mensaje es almacenado en el buzón PORENVIAR. Este queda allí pendiente hasta que el proceso DESPACHADOR lo transmita. El usuario tiene la posibilidad, mientras el mensaje esté en estado pendiente, de cancelarlo o modificarle parámetros. El texto del mensaje se maneja por separado (ver sección 2.3).

(3) El proceso ADMINISTRADOR o proceso ESCLAVO activa al proceso DESPACHADOR para que comience el despacho de los mensajes pendientes del buzón PORENVIAR.

(4) Se realiza la transferencia del mensaje a través del medio de comunicación.
(4)' Cada encabezamiento que esté marcado como certificado es depositado en el buzón ENVIADOS en espera del reconocimiento.

(5) El proceso RECEPTOR recibe todos los mensajes que llegan desde otro nodo.

(6) Todos los encabezamientos de los mensajes recibidos son depositados en el buzón RECIBIDOS.

(7) El proceso CARTERO es activado por el proceso ADMINISTRADOR o el proceso ESCLAVO para procesar la correspondencia recibida.
(7)' Los mensajes certificados son retirados del buzón ENVIADOS por cada reconocimiento recibido, dirigido a ellos.

(8) CARTERO reparte la correspondencia local a los usuarios con ayuda de MAIL. Los usuarios también son notificados por MAIL de los reconocimientos de mensajes certificados.
(8)' CARTERO deposita en el buzón NEGACKS los encabezamientos que estaban en el buzón ENVIADOS a los cuales se les ha enviado un NACK.
(8)"Los encabezamientos de mensajes con destino no local son depositados en el buzón PORENVIAR para continuar su transporte (enrutamiento). El enrutamiento se realiza mediante consultas a la tabla de destinos. Si el destino no estuviera definido, se envía un NACK al nodo origen.

(9) El usuario lee su correspondencia y las notificaciones de mensajes certificados.

El esquema de interconexión con otros correos ("gateways") que se muestra en la figura 2.1, no está incorporado en el diseño de HERMES. Sin embargo, a corto plazo se implementará un gateway para interconectar HERMES con UUCP [13].

2.2 Estructura de un Mensaje

Un mensaje HERMES está formado por un encabezamiento y un objeto (ver figura 2.2). Este modelo de estructura de un mensaje se conoce como texto-encabezamiento.

El encabezamiento contiene la información necesaria para permitir la recepción y despacho de los mensajes. El objeto es la información que el usuario desea transferir y no es interpretada para los efectos de administración y manejo de mensajes.

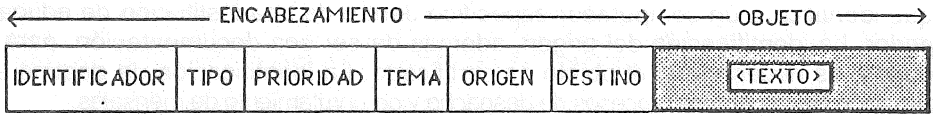


FIGURA 2.2 Estructura de un Mensaje

El encabezamiento de un mensaje HERMES está compuesto por los campos: Identificador, Tipo, Prioridad, Tema, Origen y Destino. A continuación se describe cada uno de éstos.

El campo **Identificador** asocia una identidad única a todos los mensajes residentes en un nodo. Su propósito es permitir hacer referencias del mensaje (p.ej. operaciones sobre éste por parte del usuario o envío de reconocimientos). Este identificador se usa, además, como una forma de documentación, ya que está formado por la fecha y hora en que el mensaje ha sido enviado por el usuario.

El campo **Tipo** se usa para distinguir grupos de mensajes. Existen cuatro tipos de encabezamientos: Los de tipo objeto (objetos certificados o no certificados), que tienen asociado un objeto físico y los de tipo reconocimiento (ACK o NACK), que no tienen asociado un objeto físico.

Si un objeto es certificado, el usuario del mensaje recibirá una notificación del éxito o fracaso de su llegada a destino.

Si el tipo es ACK, se trata de un reconocimento de un mensaje certificado que llegó a destino exitosamente. Si el tipo es NACK, se trata de un reconocimiento de un mensaje certificado que no llegó a destino por algún motivo (p.ej. el usuario de destino no existe).

El campo **Prioridad** sirve para jerarquizar el despacho de correspondencia. Actualmente, el sistema asigna prioridades por defecto a los mensajes; por ejemplo, un mensaje certificado tiene una prioridad mayor que un mensaje sin certificar. Estas pueden ser ajustadas por el usuario al momento de invocar el comando. El sistema de prioridades, potencialmente, puede sofisticarse asignando prioridades según grupos de usuarios y tamaños de mensajes.

El campo **Tema** es un string de caracteres, asociado al mensaje, provisto por el usuario a modo de documentación. Este es el único campo del encabezamiento no interpretado por HERMES.

Los campos **Origen** y **Destino** identifican al usuario originador y usuario destinatario del mensaje respectivamente. Cada campo está constituído por la tupla <*Nodo*, *Id-usuario* >. El campo *Id-usuario* representa al identificador del usuario en el computador (p.ej. JPEREZ). El campo *Nodo* corresponde al par <*institución*, *computador*>

lo que identifica a un computador específico dentro de una institución de educación superior. La identificación del origen, además de ser una documentación para los usuarios, es necesaria si el objeto es certificado. La identificación de destino es la información clave para los procesos de despacho y de enrutamiento de mensajes.

2.3 Sistema de Buzones y Almacenamiento de Mensajes

Dado que HERMES es un sistema de correo electrónico de almecenaje y despacho, es necesario dar una administración eficiente a los mensajes. Si los encabezamientos se manipulan por separado, se logra un manejo ágil de éstos permitiendo diferentes clasificaciones y ordenamientos. Para ello, existe un sistema común de buzones dedicados a contener exclusivamente los encabezamientos; por otra parte, existe un área particular (área HERMES) para la residencia de los objetos.

Cada encabezamiento de tipo objeto, almacenado en los buzones, apunta al lugar de residencia de su objeto asociado. Los objetos creados por el usuario, con el editor del correo, residen en área HERMES. Algunos objetos residen en los directorios de los usuarios; esto ocurre cuando el usuario escoge la opción de hacer un traspaso de un archivo que previamente existía en su directorio. En este caso, a diferencia de los demás mensajes, el usuario es dueño del mensaje y no HERMES.

Los encabezamientos de tipo reconocimiento no apuntan a objetos; éstos son sólo encabezamientos de control.

Como se mostró en la figura 2.1, HERMES maneja cuatro tipo de buzones. El buzón PORENVIAR contiene los encabezamientos de los objetos que deben salir del nodo. Estos encabezamientos pueden ser de origen local o externo. Los primeros corresponden a encabezamientos de objetos generados por usuarios locales, y los segundos a encabezamientos de objetos no locales que están en tránsito por el nodo.

El buzón ENVIADOS contiene los encabezamientos de objetos locales que efectivamente fueron despachados y que esperan ser reconocidos (objetos certificados). La existencia de este buzón permite al usuario conocer el estado de su correspondencia certificada y también efectuar un posible reenvío en caso de fracasar el traspaso a destino.

El buzón RECIBIDOS contiene todos los encabezamientos de los objetos que han llegado al nodo y que no han sido procesados.

Finalmente, el buzón NEGACKS contiene los encabezamientos de objetos locales para los cuales su traspaso a destino fracasó (mensajes que han recibido un nack). Este buzón permite al usuario efectuar un reenvío corrigiendo el tipo de error mediante algún cambio en los parámetros del comando.

3 MODELO DE COMUNICACION

HERMES ha sido modelado particionando el sistema en componentes funcionales,

cada una dedicada a un conjunto de tareas específicas. Estas componentes han sido particionadas en tres niveles, los cuales corresponden a una simplificación del modelo de referencia de comunicaciones ISO. A continuación se describe cada uno de estos niveles.

3.1 Nivel De Aplicación

Este nivel, también llamado agente del usuario (AU), es el nivel más alto. Interactúa con el usuario o, de otro modo, es la interfaz entre el usuario y los servicios de despacho y recepción. Las componentes de este nivel asumen todas las funciones que están directamente bajo control de una persona, actuando en favor de ella, asistiéndola en el envío y lectura de mensajes. El AU permite una comunicación persona a persona muy parecida a la que se lograría con un sistema de correo convencional.

Este nivel provee un conjunto de comandos para atender los requerimientos del usuario. Estos comandos son los siguientes:

- Enviar: El usuario puede crear (editar) y enviar un mensaje o puede enviar un archivo ya preparado en su directorio.

- Reenviar: El usuario puede cambiar los parámetros de un mensaje que no ha sido despachado aún o cuyo traspaso a destino fracasó.

- Buzón: El usuario puede listar los encabezamientos de los mensajes que están por despacharse, los que ya han sido despachados y esperan ser reconocidos o aquellos que han recibido un NACK.

- Cancelar: Es posible cancelar un mensaje que aún no ha sido despachado o que ha recibido un NACK.

Una vez efectuado un comando, este nivel procede a ajustar los parámetros para los requerimientos del nivel de transporte, utilizando el sistema de buzones. Esto incluye la definición del encabezamiento del mensaje, el depósito de éste en el buzón correspondiente y el almacenamiento del objeto si es necesario. Además, este nivel se encarga de entregar la correspondencia al usuario para que éste pueda leerla.

Con respecto al reparto de mensajes que llegan, el sistema realiza una interacción directa con el correo local incorporado en cada máquina (MAIL), el cual finalmente entrega la correspondencia a los usuarios. De ese modo, el usuario puede mirar la correspondencia HERMES de la misma manera que normalmente lo hace mediante su correo local. Este esquema se usa para el reparto, tanto de mensajes como de archivos. Luego, desde el punto de vista de la entrega de correspondencia, no existe diferencia entre ambos tipos de objetos.

3.2 Nivel De Transporte

La función exclusiva de las componentes del nivel de transporte es permitir, con la ayuda del nivel de comunicación, el despacho y la recepción de mensajes entre dos nodos, estableciendo la sincronización necesaria de los procesos de despacho y recepción

entre dos nodos. Además, procesa los mensajes recibidos, incluyendo enrutamiento y reparto de mensajes a usuarios.

Las componentes de este nivel son básicamente, el sistema común de buzones, el área de almacenamiento de los mensajes y las tablas de destinos y líneas que facilitan el enrutamiento de mensajes y el establecimiento de sesiones de comunicación entre nodos adyacentes. Una sesión consiste en la conexión remota entre dos nodos (login remoto), la transferencia de correspondencia entre ambos y, finalmente, su desconexión.

El despacho y recepción de mensajes, se realiza, en forma automática, mediante sesiones periódicas establecidas entre dos nodos. En una sesión hay un nodo que contiene el proceso administrador, que actúa como nodo maestro y el otro es el nodo esclavo. El nodo esclavo es activado por el nodo maestro cada vez que este último decide establecer una sesión. Esto significa que el traspaso de mensajes en el nodo esclavo depende directamente del nodo maestro. Es posible descentralizar la responsabilidad haciendo coexistir en cada nodo el maestro y el esclavo. Con la existencia simultánea de ambos, la iniciativa de los trapasos de mensajes se reparte desde el momento en que ambos nodos pueden decidir establecer una sesión, pero manteniendo la relación maestro-esclavo en una sesión cualquiera.

El administrador tiene a su cargo todas las líneas de los nodos vecinos, las que recorre permanentemente para establecer sesiones. En cada sesión, el administrador sincroniza su proceso DESPACHADOR con el proceso RECEPTOR del esclavo para vaciar su buzón PORENVIAR en el buzón RECIBIDOS del nodo esclavo. A continuación se establece el sincronismo para los procesos RECEPTOR del nodo maestro y DESPACHADOR del nodo esclavo. Así, el esclavo vacía su buzón PORENVIAR traspasando la información al buzón RECIBIDOS del maestro. Una vez que el administrador ha recorrido todas las líneas, procede a repartir o enrutar la correspondencia recibida mediante el proceso CARTERO. El esclavo, realiza el reparto y enrutamiento en su nodo, inmediatamente finalizada su sesión particular.

Este esquema maestro-esclavo es necesario por razones de eficiencia y de disponibilidad restringida de líneas. Para el acceso y liberación de líneas se manejan tres parámetros por línea: El horario disponible de la línea, el número de intentos fallidos en el acceso dentro del horario y el horario de la próxima visita a la línea (ver figura 3.2).

Las líneas están normalmente disponibles sólo a ciertas horas del día, de modo que los traspasos por una línea deben regirse por su horario de disponibilidad. Si el número de intentos fallidos en usar una línea excede un cierto límite, se notifica en un archivo bitácora y la línea es ignorada en los próximos accesos. En cuanto al horario de la próxima visita, después de terminar una sesión, se espera un cierto lapso de tiempo (p.ej. 1 hora) para intentar la próxima sesión, liberando el recurso durante ese lapso.

El administrador también se rige por un horario; no es adecuado que éste opere en horas de un flujo nulo de mensajes, aún en horas disponibles de alguna línea. Los horarios del administrador y de las líneas son particulares de cada institución.

Actualmente, el programa administrador está implementado en una máquina

UNIX y el programa esclavo en una máquina VMS. Con ello se provee una amplia cobertura de comunicación a nivel universitario dada la gran disponibilidad de estos equipos en las universidades.

Figura 3.2 Esquema Administrador Esclavo

3.3 Nivel de Comunicación

Este nivel es el encargado de establecer la transferencia de mensajes, libre de errores, a través del medio de comunicación. Este nivel es básicamente una adaptación y automatización del protocolo Kermit. La gran disponibilidad de este protocolo en diversas máquinas motivó su adaptación, lo que facilita la portabilidad del nivel de comunicación de HERMES a otras máquinas.

El traspaso de mensajes se realiza mediante los procesos DESPACHADOR y RECEPTOR. Estos procesos automatizan la transferencia de grupos de mensajes dentro del protocolo Kermit, y son activados por el proceso administrador una vez que el nodo maestro ha establecido la conexión con el nodo esclavo. Esta conexión equivale a un "login remoto" desde el nodo maestro. Cada nodo administrador hace uso de tablas que contienen los procedimientos para establecer conexiones con sus nodos vecinos.

4. CONCLUSIONES

Hermes es una sistema de correo electrónico distribuído que provée las operaciones básicas de despacho y recepción de mensajes usuario a usuario. Su objetivo principal es proporcionar un servicio básico de intercambio de información entre equipos heterogéneos tal como se presenta en el ámbito interuniversitario.

En lo que respecta al nivel de comunicación entre nodos, el protocolo de transferencia de mensajes está basado en el protocolo Kermit. Esto, además de asegurar la confiabilidad de las transmisiones, facilita la portabilidad del correo hacia diversos computadores.

Hermes provée un editor básico de texto, mediante el cual los usuarios pueden crear mensajes (archivos ASCII) los cuales conforman la correspondencia básica de

Hermes. Además, permite el envío de archivos previamente creados en el directorio del usuario (p.ej. un archivo de datos). Sin embargo la distinción entre mensajes y archivos no es transparente para el usuario, ya que el recipiente de destino de ambos es el mismo.

El despacho de correspondencia puede ser certificado. Esto significa que el usuario puede pedir la notificación del éxito o fracaso de la llegada a destino de ciertos mensajes.

El transporte de mensajes entre un nodo y otro es completamente transparente para el usuario. El sistema establece sesiones de comunicación entre dos nodos adyacentes, en forma automática, y provee mecanismos para realizar discado y respuesta automática entre nodos remotos.

El despacho de correspondencia se realiza mediante un esquema de intervalos de tiempo, que es paramétrico por cada línea de comunicación. La frecuencia de verificación del acceso a cada línea depende del horario del administrador del nodo. Estos valores pueden ser ajustados externamente. El programa administrador verifica el comportamiento de cada línea, y puede desactivar automáticamente una línea si ésta presenta fallas reiteradas en la transferencia de mensajes.

Cada nodo opera en la modalidad de "store and forward". Esto es, parte de la correspondencia recibida por un nodo puede estar constituída por mensajes "en tránsito" hacia otro nodo. Los mensajes recibidos cuyo destino no es el nodo local, son depositados en el buzón de salida, previa verificación de la existencia del nodo destino.

No existe en el nivel de aplicación (agente usuario), una integración entre la generación y lectura de la correspondencia. Ambas funciones se realizan en forma independiente. Actualmente, se utiliza el correo local de cada nodo para llevar finalmente la correspondencia al usuario. Esto limita la funcionalidad del correo electrónico, en cuanto al redireccionamiento de mensajes. Sería conveniente una interacción más directa con el correo local para permitir anotaciones adicionales en el mensaje, por parte del usuario, para ser posteriormente reenviado, al usuario de origen (respuesta) o a otro usuario ("forwarding").

HERMES corresponde a una primera etapa en el desarrollo de un sistema de correo electrónico interuniversitario. En el futuro se pretende ampliar su funcionalidad, seguridad y robustez (p.ej. incorporar un nivel de presentación que permita facilidades de codificación de la correspondencia), para incorporarlo como una aplicación de una futura red de computadores para las instituciones de educación superior. Una preocupación a corto plazo es la de implementar sistemas de interconexión ("gateways") de correos electrónicos para permitir, tanto la comunicación de correos incompatibles (p.ej. HERMES-UUCP) como el acceso a servicios de correo electrónicos públicos (p.ej. TELEMAIL).

BIBLIOGRAFIA

[1] R. A. A. BUHR, D. A. MACKINNON; "Mailroom: A Computer-Based Message System Model for Person-to-Person and Process-to-Process Communication". The 3rd International Conference on Distributed Computing Systems. Miami/Ft. Lauderdale, Florida. October 18-22, pp. 818-823, 1982.

[2] DOUGLAS COMER, JOHN T. KORB; "CSNET Protocol Software: The IP-to-X.25 Interface". SIGCOMM'83 Symposium, Communications Architectures & Protocols, University of Texas at Austin, pp. 154-159, March 8 & 9, 1983.

[3] IAN CUNNINGHAM; "Electronic Mail Standards to Get Rubber-Stamped and Go Worldwide". Data Communications, May 1984, p. 159.

[4] FRANK DA CRUZ, BILL CATCHINGS; "Kermit: A File Transfer Protocol for Universities. Part 1: Design Considerations and Specifications". Byte, p. 255, June 1984.

[5] FRANK DA CRUZ, BILL CATCHINGS; "Kermit: A File Transfer Protocol for Universities. Part 2: States and Transitions, Heuristic Rules and Examples", Byte, p. 143, July 1984.

[6] PETER J. DENNING, ANTHONY HEARN, C. WILLIAM KERN; "History and Overview of CSNET". SIGCOMM'83 Symposium, Communications Architectures & Protocols, University of Texas at Austin, pp. 138-145, March 8 & 9, 1983.

[7] ISO TC97/SC16 "Open System Interconection - Basic Reference Model, (ISO Draft proposal 7498)". Computer Communication Review, pp 15-65, April 1981.

[8] WARWICK S. FORD; "Portable Implementation of Network Architecture Layers". SIGCOMM'83 Symposium, Communications Architectures & Protocols, University of Texas at Austin, pp. 240-245, March 8 & 9, 1983.

[9] JOSE GARCIA-LUNA-ACEVES, FRANKLIN F. KUO; "Dessign Issues of Protocols for Computer Mail". Seventh Data Communications Symposium -1981. Maria Isabel Sheraton Hotel Mexico City, Mexico. October 26-29, pp. 28-36, 1981.

[10] ROBERT GRAFF; "Modeling an Electronic Mail Network: A Primer". Data Communications, p. 163, August, 1983.

[11] L. LANDWEBER, M. LITZKOW, D. NEUHENGEN, M. SOLOMON; "Architecture of the CSNET Name Server". SIGCOMM'83 Symposium, Communications Architectures & Protocols, University of Texas at Austin, pp. 146-153, March 8 & 9, 1983.

[12] LOH-PING YU; "The Anatomy of a Distributed Electronic Mail Network". Data Communications, p. 153, March 1985.

[13] D. A. NOWITZ; "UUCP Implementation Description". Ultrix-32 Supplementary Documents, Vol. 3.

[14] PETER SCHICKER; "The Computer Based Mail Environment - An Overview". Computer Networks 5, pp. 435-443, 1981.

[15] GERALD TOMANEK; "Implementing Electronic Mail in a Telephone System: More Than Just Talk". Proceedings, National Computer Conference, pp. 527-531, 1980.

DATALEGIS: LA DESMITIFICACION DE LA INFORMATICA JURIDICA DOCUMENTARIA

José Lucas Dugand Pinedo
Bogotá - *Colombia*

CREACION DE DATALEGIS: HISTORIA Y MOTIVOS

Durante los últimos años, el uso de los computadores en campos especializados como la ingeniería, la medicina, arquitectura, etc., se ha vuelto algo normal. El empleo de estas herramientas por parte de profesionales de estas ramas ha llegado a ser idispensable para lograr un nivel de competitividad adecuado y se ha convertido en un factor de superviviencia en el mundo de los negocios.

Una de las últimas ramas profesionales en aprovechar el poder del computador ha sido la del derecho. La razón es muy simple: Los abogados constituyen (en términos generales) uno de los gremios más difíciles en cuanto a la aceptación de los avances tecnológicos y más si esa tecnología es aplicada a su campo de acción. El común denominador del por qué de los fracasos de la mayoría de proyectos de Informática Jurídica en nuestros países ha sido precisamente el hecho de que no ha sido posible conciliar los intereses y los puntos de vista de los abogados con los ingenieros de sistemas.

Un estudio realizado en Colombia acerca de los proyectos sobre Informática Jurídica que habían fracasado demostró que en el 100% de los casos el motivo principal de ese fracaso había sido la imposibilidad de encontrar puntos en común entre dos ciencias diferentes desde todo punto de vista. Este mimo estudio reveló que el inmenso costo debido a los grandes equipos de computación necesarios para llevar a cabo el proyecto, era otro de los motivos de fracaso, así como la terminología poco amigable de los programas, cosa inaceptable para el hombre de leyes.

Estos mismos puntos fueron reafirmados en varios eventos y seminarios, tales como el I Congreso Iberoamericano de Informática Jurídica realizado en 1984 en República Dominicana, donde con asombro vimos que eramos los únicos representantes de la Empresa Privada. En dicho evento se admitió, casi de forma unánime, que el único ente capaz de llevar a cabo proyectos de Informática Jurídica documentaria debido a los altos costos, era el Estado.

Con estas hipótesis en contra, un grupo de ingenieros de sistemas de LEGIS, la Empresa Editorial más grande de Latinoamérica en materia de información legal, con cerca de 110.000 Empresas suscritas a un sistema de información actualizable, se dió calladamente a la tarea de trabajar en un pre proyecto que se denominó DATALEGIS.

CONCEPCION DE UN PROYECTO

Tomando como experiencia los puntos débiles de otros proyectos fracasados y manteniéndonos al tanto de los éxitos en otros países, el grupo de Ingeniería de Sistemas de LEGIS se dió a la tarea de desarrollar, antes que un proyecto, una filosofía que perdurara y se mantuviera como bandera durante el desarrollo de todo el proyecto. Los objetivos, que al mismo tiempo nos sañalaban las dificultades y obstáculos que debíamos superar, eran los siguientes:

1. El proyecto debía ser dirigido por profesionales del derecho, conservando a Ingeniería de Sistemas como soporte fundamental y puente entre lo jurídico y lo técnico.

2. El software debería ser desarrollado para una máquina de tipo medio, desbaratando de esta forma el mito de los altos costos y máquinas exagerádamente grandes.

3. El software debería estar orientado al usuario final, en este caso el abogado. En lo posible debería estar desprovisto de terminología Técnica, comandos en inglés y cosas por el estilo que invitan al usuario potencial a rechazar el sistema desde el comienzo.

4. La consulta debería estar a la altura del lenguaje natural, logrando efectuar la búsqueda de los textos de una forma simple en el mismo lenguaje y terminología utilizado por el abogado.

5. El precio de la consulta debería ser bajo, logrando de esta forma la divulgación masiva y la democratización de la Información Jurídica.

6. Posibilidad de interacción con otras bases de datos legales, permitiendo la comparación y el análisis de información proveniente de diversas fuentes.

7. Base de datos "Full Text" conteniendo no sólo las normas legales sino la jurisprudencia, doctrina, bibliografía, ejemplos, etc.

Definidas estas premisas y una vez lograda la identificación clara de los objetivos, se inició la tarea del análisis y diseño del sistema, hace aproximadamente dos años.

DESARROLLO DEL PROYECTO

CONFORMACION DEL EQUIPO DE TRABAJO

Con el ánimo de cumplir fielmente con la filosofía que había originado DATALEGIS, se nombró un equipo de trabajo para dirigir el proyecto. Este equipo estuvo integrado por los abogados GERARDO ARENAS y HECTOR PRIETO, redactores del Régimen Laboral Colombiano y del Régimen Tributario respectivamente y el Ingeniero JOSE LUCAS DUGAND por la parte técnica; se conformó igualmente un Comité de Informática Jurídica integrado por las personas antes citadas y por la Presidencia de la Compañía, órgano encargado de asesorar a los directores del proyecto durante todas sus etapas.

La conformación del equipo de trabajo fue uno de los factores más importantes para el éxito del programa, ya que los mismo abogados encargados de compilar, analizar y redactar obras de tipo legal, fueron los encargados de hacer ahora lo mismo, pero por medios electrónicos. Alquien definió entonces a DATALEGIS, dentro del contexto de nuestra Empresa, como "un cambio de medio, más no de producto".

El grupo de abogados, se dió a la tarea de compilar y organizar toda loa información legal en fichas adecuadas para la grabación de textos, agrupándolas por temas de forma tal que se facilitara la asignación de claves y descriptores. Las palabras claves son las más pequeñas unidades lexicales o elementos significativos dentro de un documento; generalemente constituye una palabra tomada de su propio contexto u otra fuera del contexto pero que expresa con más propiedad un significado (sinónimo jurídico). La asignación de estas palabras claves puede llegar a ser la base del éxito en la consulta.

SELECCION DEL HARDWARE

De acuerdo con los objetivos del proyecto de lograr una consulta económica para el usuarios, el programa debía adecuarse a una estructura de costos muy estricta. Una vez desechados equipos grandes como la serie 4300 de IBM, Hewlett Packard 3000 o WANG US/100, cuyos altos costos se verían innecesariamente incrementados con software tipo "Stairs" para búsqueda de textos, nos inclinamos por un S/36 de IBM, máquina mediana con grandes posibilidades de ampliación. El hecho de haber llegado a ser en tan sólo tres años el minicomputador más vendido en la historia de la computación, influyó poderosamente en la decisión ya que garantizaba una continuidad en el servicio por muchos años sin cambios en el

software.

DESARROLLO DEL SOFTWARE

La ausencia de software especializado tipo "Stairs" para búsqueda de textos en el S/36 fue el primer problema encontrado, pero al mismo tiempo fue el hecho que nos impulsó a desarrollar una base de datos propia, encaminada a obtener una consulta ágil y amigable. Utilizando el lenguaje RPG II bajo el sistema operacional SSP y algunas rutinas en lenguaje de máquina se diseñó DATALEGIS, con características muy propias en cuanto a la eliminación total de términos técnicos carentes de sentido para el abogado pero al mismo tiempo con una adaptabilidad muy grande que permite la creación de bancos de datos con la legislación de cualquier país. A manera de prueba hemos grabado el capítulo de salarios del Régimen Laboral Venezolano sin hacer ningún cambio en el software excepto la terminología jurídica propia, obteniendo resultados ampliamente satisfactorios.

En cuanto a la rapidez de la consulta, después de amplias investigaciones del grupo de ingeniería, se desarrolló un algoritmo de búsqueda que proporciona un tiempo de respuesta nunca mayor de cinco segundos, sin importar lo extenso del tema. Igualmente se ha trabajado sin cesar en continuas mejoras a la "amigablidad" del software, orientándolo al usuario final; hemos eliminado todo término técnico y comandos en inglés evitando de esta forma el rechazo inicial por parte del abogado. Todo lo que éste necesita es una pequeña introducción en el manejo del teclado.

LENGUAJE NATURAL

La consulta es efectuada mediante una o varias palabras claves o descripciones, conformando un lenguaje jurídico muy cercano al lenguaje natural. Como señalamos anteriormente, un extenso análisis de la parte legal permitió asignar claves dentro y fuera del contexto del documento, lo cual facilita la recuperación del texto por palabras o temas aunque éstos no figuren expresa y físicamente dentro del mismo.

El software guía al usuario, llevándolo inicialmente a las normas legales que coincidan con el tema solicitado. Una vez seleccionadas estas normas, le permite descubrir un "universo de información jurídica" referente a ellas como es la jurisprudencia, doctrina, bibliografía, ejemplos, normas concordanres, etc. Si el usuario tiene una impresora conectada a su micro, puede solicitar la impresión de esta

información.

COSTO DE LA CONSULTA

Uno de los aspectos más importantes en una base de datos pública de cualquier especie es la divulgación masiva de la información. Más aún en el caso de la información jurídica, por lo general limitada a la interpretación de unos cuantos; las bases de datos legales democratizan la información jurídica y es por esta razón que LEGIS ha puesto especial empeño en reducir los costos para lograr llegar a la mayor cantidad de usuarios posible. Es así como DATALEGIS ha salido al mercado con un precio de consulta de 25 centavos de dólar por minuto, constituyéndose en el banco de datos más económico en su tipo, llegando a ser hasta diez veces más barata la consulta que en los Estados Unidos a pesar de pagar más cara la tecnología necesaria.

Es importante anotar que aquellos usuarios que posean impresora conectada a su terminal, pueden hacer uso de la opción de imprimir las normas legales, jurisprudencia o doctrina sin cargo adicional.

Aquellos usuarios en ciudades remotas, fuera de Bogotá, deben obviamente asumir los costos correspondientes a las llamadas telefónicas de larga distancia para conectarse a la base de datos, haciendo la salvedad de que en Colombia pronto dispondremos de una red pública de datos con lo cual se minimizará este problema.

INTERACCION CON OTRAS BASES DE DATOS

Antes de terminar el presente año, DATALEGIS ofrecerá al público la posibilidad de consultar otras bases de datos que tengan alguna relación con la parte legal. Es así como, por ejemplo, se está trabajando para introducir en el sistema las estadísticas y encuestas salariales por sector y actividad económica, cuya fuente es ACRIP, la Asociación Colombiana de Relaciones Industriales. A esta información tendrán acceso sin costo adicional los usuarios de la base de datos de Legislación Laboral, constituyéndose en un servicio complementario muy importante.

Por otro lado, ya se encuentra disponible para empresas grandes que tengan sindicato y convenciones muy extensas y complejas, la posibilidad de integrar la Legislación Laboral con los pactos o convenciones colectivas de trabajo. De esta forma una Empresa o Sindicato puede consultar en pantalla qué dice la Legislación Laboral acerca de una determinado tema y

qué dice su pacto o convención colectiva sobre ese mismo tema.

De obtenerse el permiso correspondiente de parte de las Empresas que involucren sus convenciones en las bases de datos, es posible que esta información se haga pública con lo cual Empresas, Sindicatos y el mismo legislador se beneficiarían del cruce de la información enriqueciendo la Legislación Laboral.

BASE DE DATOS "FULL TEXT"

La mayoría de bases de datos jurídicos presentan al usuario resúmenes o "abstracts" de la información legal debido a las limitaciones de máquina. Hemos entendido que esta limitación no debe existir en un banco de datos público, razón por la cual el texto que se ha incorporado es completo. A manera de ejemplo, la base de datos laboral es cinco veces mayor en información y contenido que el tradicional Régimen Laboral Colombiano impreso en libro de hojas sustituibles. Este amplio panorama sumado al hecho de la exhaustividad en la búsqueda que ofrece el software, permite análisis más profundos sobre un determinado tema legal, facilitanto así la labor de jueces y abogados.

ASPECTOS TECNICOS DE DATALEGIS

EQUIPO	:	IBM S/36
LENGUAJE	:	RPG II
SISTEMA OPERACIONAL	:	SSP RELEASE 3.0

DISEÑO

El lenguaje escogido para diseñar la base de datos fue el RPG II. Aunque lo importante de una base de datos es su diseño y concepción, independientemente del lenguaje utilizado. El RPG II y especialmente el RPG III hacia el cual tiende el S/36 con las actualizaciones de sistema operacional que viene haciendo regularmente IBM, cuenta con poderosas herramientas y estructuras que hicieron posible diseñar un proyecto flexible que aprovechara toda la arquitectura de múltiples procesadores del S/36. La lógica estructurada de este lenguaje, propia del mismo, hizo posible lograr un banco de datos "Full Text" a pesar de no ser un procesador de palabra propiamente dicho, permitiendo la creación, corrección y mantenimiento de los textos en forma interactiva y amigable. La rapidez de proceso

permite igualmente compartir recursos a velocidades apropiadas, con lo cual se logra una consulta ágil a través de cualquiera de las ocho líneas telefónicas disponibles.

Uno de los problemas propios de toda base de datos jurídica es el relativo a los sinónimos. Cualquier procesador localiza un texto por medio de una palabra que "físicamente" se encuentre dentro de él, pero el problema viene cuando buscamos a través de un sinónimo: El procesador normal no encuentra por sinónimos, problema que se puede considerar grave en el aspecto jurídico. DATALEGIS ha sido concebido de tal forma que evita este problema a través de múltiples palabras claves sinónimas que encuentran el texto correspondiente aunque no esté físicamente incorporada la palabra en él.

COMUNICACIONES

Es tal vez la parte más complicada en el diseño de DATALEGIS, porque el S/36 no es lo que podría llamarse un "Main Frame" o equipo central para un ambiente de comunicaciones. El primer problema por solucionar eran las comunicaciones sincrónicas, ya que estas eran las únicas soportadas por el S/36 (a partir del release 4.0 este problema queda resuelto con un adaptador de comunicaciones asincrónicas). Esto implicaba que el posible usuario de la base de datos se viera abocado a adquirir una costosa tarjeta de comunicaciones sincrónicas y un modem telefónico sincrónico también con un alto costo. Para resolver este problema LEGIS adquirió unos convertidores de protocolo sincrónico/asincrónicos para cada línea telefónica, facilitando así al usuario la comunicación ya que todo lo que requiere, aparte obviamente de un microcomputador, es un modem asincrónico de bajo costo (se consiguen desde US$300) y un software para comunicarse y emular un terminal de un S/36, que es proporcionado al usuario por LEGIS.

Ese software requiere para ser ejecutado que el microcomputador utilizado sea IBM o compatible.

Los MODEMS receptores adquiridos por LEGIS para recibir la comunicación externa de los usuarios, son totalmente automáticos y se gradúan de acuerdo con la velocidad de transmisión del modem del usuario (desde 300 hasta 7400 baudios).

En resumidas cuentas, en DATALEGIS están conjugados los últimos adelantos tecnológicos en cuanto a diseño, procesadores, comunicaciones y seguridad.

La conclusión más importante a que podemos llegar es que la contibución que ha hecho este proyecto colombiano a la Informática Jurídica Latinoaméricana es amplio, ya que podemos

concluir que no necesitamos de costosísimos equipos ni programas importados de otras latitudes. La realidad de una base de datos jurídica con la legislación colombiana, realizada a costos razonables y con recursos e ingeniería colombiana, es una invitación a los demás países latinoamericanos a continuar por ese camino, el cual nos llevará en un futuro muy próximo a tener bases de datos latinoamericanas en una misma máquina logrando de esta manera efectuar lo que los abogados denominan "el derecho comparado" entre naciones hermanas.

El Dr. Edgar Salazar Cano de la Universidad de Carabobo en su libro "Instrucción Programada y Documentación Automática" señala: "La Informática Jurídica y el Derecho Cibernético se encuentran hoy no en los límites sino en el comienzo de nuevas posibilidades. El tener en cuenta dichas posibilidades es un tema actualísimo de la teoría y la práctica del derecho. El futuro enseñará lo que hace falta. En todo caso las objeciones de principio en contra de estas disciplinas han dejado de ser puntos de vista racionales".

Es verdad que tan sólo estamos viendo la punta del iceberg. Las posibilidades que se abren tanto a abogados, legisladores y jueces son inmensas, ya que del análisis exhaustivo de la información jurídica por medios electrónicos, se derivará beneficios mayores en el campo de lo legal para todas las naciones del área.

ACCESO DIRECTO EN MENUS

Mario Jofré - José A. Pino
Universidad de Chile
Santiago - *Chile*

Resumen

El acceso directo a los servicios o acciones de un sistema de software ha sido sugerido como una manera de mejorar las interfaces de menús. Este trabajo describe un experimento para evaluar el desempeño de los usuarios cuando se agrega la posibilidad de acceso directo a una interfaz tipo menú. Los resultados de este experimento muestran que esta mejora es efectivamente útil.

1. INTRODUCCION

Una gran variedad de sistemas de software poseen en la actualidad interfaces tipo menú para interactuar con el usuario. Estas interfaces son muy útiles para usuarios con poca experiencia en el uso del sistema en particular, y de sistemas de software en general, pues permiten usar un sistema sin tener un conocimiento previo de su funcionamiento.

Un menú consiste en una lista de opciones presentadas al usuario en la pantalla de su terminal; la persona escoge una de esas opciones ingresando un código numérico o alfabético, o bien, moviendo el cursor mediante un dispositivo "apuntador". Si la opción escogida no es una acción o servicio directo del sistema, se presenta una nueva pantalla con opciones más específicas, y el proceso continúa. Eventualmente, el usuario debe escoger alguna acción o servicio terminal. Durante esta comunicación, el usuario puede haber ingresado valores de parámetros u otra información antes de seleccionar una acción del sistema.

La relación existente en los menús entre opciones generales y opciones más específicas puede ser asociado al concepto de árbol de decisiones. El primer nivel del menú corresponde a la raíz y sus hijos. Cada pantalla que conduzca a opciones más específicas se asocia a un nodo interno del árbol, y estas opciones serán sus hijos. Las pantallas que corresponden a acciones o servicios propios del sistema, esto es, que no conducen a opciones más específicas, son las hojas del árbol. Usaremos esta terminología de ahora en adelante.

La utilización de menús también tiene problemas, los que se acrecientan cuando la interfaz es grande, es decir, cuando la altura del árbol de decisiones correspondiente tiene muchos niveles (por ejemplo, seis

o más). Uno de estos problemas es el llamado problema de la Conciencia del Estado, identificado por Apperley y Spence [1], y al cual se han propuesto varias soluciones [6]. Otro problema importante, y en el cual nos concentraremos en este trabajo, ocurre cuando la interfaz de menús se transforma en una molestia más que en una ayuda [2]. Naturalmente, esto no ocurre con los usuarios ocasionales del sistema, sino con quienes van adquiriendo experiencia con el sistema o con los que lo utilizan constantemente (usuarios regulares). En ambos casos, después de un período de aprendizaje, la persona sabe perfectamente cuáles son los pasos que debe dar para lograr algo del sistema, y la o las opciones específicas dentro del sistema que sirven para este propósito. En estas circunstancias, la interfaz misma se transforma en un obstáculo que quita tiempo. Para el usuario experto se agregan además otros factores, como por ejemplo, la falta de sensación de control sobre el sistema, que es un punto importante para él [3].

Para solucionar el problema anterior se han sugerido algunas formas de acceso rápido a los nodos (acceso directo). Las tres formas principales son las siguientes [4]:

(1) como alternativa a ingresar un código para seleccionar un nodo del siguiente nivel, se puede ingresar una secuencia a la vez, la cual representa un camino válido desde el nodo actual hasta un sucesor de éste.

(2) una forma de optimizar el proceso anterior es dar nombres a secuencias de caminos más usados. Luego, cuando se requiere una secuencia dada, se invoca este nombre, el cual expande la secuencia requerida.

(3) una última alternativa es dar nombres a los diferentes nodos. Luego, si se desea llegar a un nodo en particular desde cualquier nivel, basta con invocar su nombre.

2. PROPOSICION

Basados en la tercera alternativa, y tomando en cuenta las necesidades de los usuarios expertos y regulares, parece interesante combinar la idea de interfaz tipo menú y aquellas de tipo comando.

Esto se logra dando nombres sólo a los nodos terminales, esto es, a las acciones o servicios reales del sistema. La interfaz tipo menú permanece inalterable. Luego, un usuario novicio en el sistema sólo sabrá de la existencia de la interfaz tipo menú. Un usuario con más experiencia

debería aprender, a medida que usa el sistema, a utilizar estos nombres, si no en su totalidad, al menos aquellos que usa en forma regular.

Una forma de informar los nombres definidos para estos nodos, es mostrarlos en la pantalla, en algún lugar específico, cada vez que se invoca una opción asociada a un nodo terminal.

Para tener una idea del comportamiento de los usuarios frente a sistemas de este tipo, se hizo un pequeño test que se explica a continuación.

3. PRUEBAS

El experimento realizado consistió en comparar el desempeño de los usuarios en términos del tiempo que demoraban en cumplir una cierta tarea, utilizando un sistema tradicional con menús versus un sistema de menús que incluye claves de acceso a las hojas del árbol de decisiones.

Los sujetos fueron estudiantes de primer año de ingeniería de la Universidad de Chile, con poca o ninguna experiencia en computadores. Debieron usar un sistema de información para contestar un cuestionario que contenía preguntas sobre proyectos de investigación desarrollados en la Universidad. Las materias incluidas no fueron triviales, de modo de evitar que las respuestas fueran dadas sin consultar el sistema, pero tampoco eran complejas (ver Anexo).

El sistema de consultas tenía una interfaz tipo menú de cinco niveles, e incluía instrumentación para registrar tanto las selecciones realizadas por cada estudiante, como el tiempo que demoraba en hacerlas.

Los sujetos fueron particionados en dos grupos: uno llamado grupo EXPERIMENTAL (GE), el cual contaba con una interfaz de menú, incluyendo claves para las hojas, mientras que el segundo grupo llamado de CONTROL (GC), usó una interfaz común sin estas claves. Ambos grupos recibieron una breve explicación del uso del sistema; además los del Grupo EXPERIMENTAL fueron informados de la existencia y uso de las claves.

El cuestionario se dividió en dos etapas, la primera, considerada de aprendizaje, consistió de 15 preguntas, la mayoría referidas a información contenida en tres de las hojas, para simular la existencia de usuarios regulares. En la segunda etapa, llamada de evaluación, contuvo 8 preguntas, 5 de las cuales fueron referentes a los mismos tres nodos. En ella se registraron las selecciones realizadas y los tiempos tomados en hacerlas. Un porcentaje de las respuestas dadas por los sujetos fueron rechazadas para el análisis estadístico, por razones tales como respuesta

incompleta al cuestionario, o no realizada en el orden fijado previamente. En total, del grupo EXPERIMENTAL fueron consideradas las respuestas de 18 estudiantes, mientras que en el de CONTROL se consideraron 16.

En la figura No. 1 se grafica el tiempo medio tomado por los individuos de ambos grupos en el recorrido del árbol de decisiones desde el nodo actual hasta el nodo que se necesita para contestar la siguiente pregunta. La tabla No. 1 muestra la desviación estándar asociada a los tiempos medios. Las preguntas 1, 2, 4, 5 y 7 se refieren a los nodos mencionados anteriormente (los más visitados).

4. RESULTADOS Y ANALISIS

De una inspección visual de tablas y gráficos se observa un comportamiento disímil entre ambos grupos. Para las preguntas 1, 2, 4, 5 y 7, el comportamiento se muestra superior para el grupo EXPERIMENTAL. En las preguntas 3 y 8 esta situación se invierte, mientras que en la pregunta 6 el resultado se ve similar.

Se recurrió a un test de Student para el análisis estadístico de cada pregunta por separado. También, se hizo un test multivariado de Hotelling [6] para analizar la significancia estadística de todas las preguntas consideradas en su conjunto.

El test de Hotelling muestra que, con una confianza del 95%, los grupos son distintos, sin especificar en qué difieren.

Un examen de los t de Student, muestra que esta diferencia significativa se debe esencialmente a la pregunta 7, en que de acuerdo a lo esperado, el comportamiento del grupo EXPERIMENTAL es mejor que el del grupo de CONTROL (valor-p = -0.001).

Por otro lado, los signos de la diferencias para las preguntas 1, 2, 4 y 5 se comportan de acuerdo a lo previsto, aún cuando ellas no alcanzan a ser significativas. Lo mismo se puede aplicar a la pregunta 6.

En las preguntas 3 y 8, el signo está invertido con respecto a las otras preguntas, sin embargo, la diferencia no alcanza a ser significativa.

Las preguntas 1, 2, 4, 5 y 7 son, como ya se dijo, preguntas que pueden ser respondidas con la información contenida en las hojas ya visitadas del árbol de decisiones. Dado que a los sujetos del grupo EXPERIMENTAL se les informó en la primera etapa de la existencia y uso de las claves, se supone que la mayoría de ellos usó esta opción, o al menos lo intentó. De este modo se explica el comportamiento superior en esas preguntas, ya que el

Tabla No 1

Tabla de tiempos medios por pregunta por grupo					
Pregunta	Experimental		Control		T-student
	tiempo [s]	Desv. Std.	tiempo [s]	Desv. Std.	Valor-p
1 *	35.27	13.4	46.67	33.1	-0.227
2 *	29.44	27.0	39.53	25.0	-0.275
3	25.50	21.8	13.47	14.9	+0.071
4 *	34.06	24.7	44.07	22.7	-0.235
5 *	32.17	47.7	39.07	18.8	-0.578
6	53.44	22.1	55.27	34.2	-0.860
7 *	20.06	9.9	46.33	24.6	-0.001
8	21.00	15.3	16.00	7.5	+0.232
T-Hotelling Valor-p= 0.5					

*: Nodos frecuentemente visitados en forma previa.

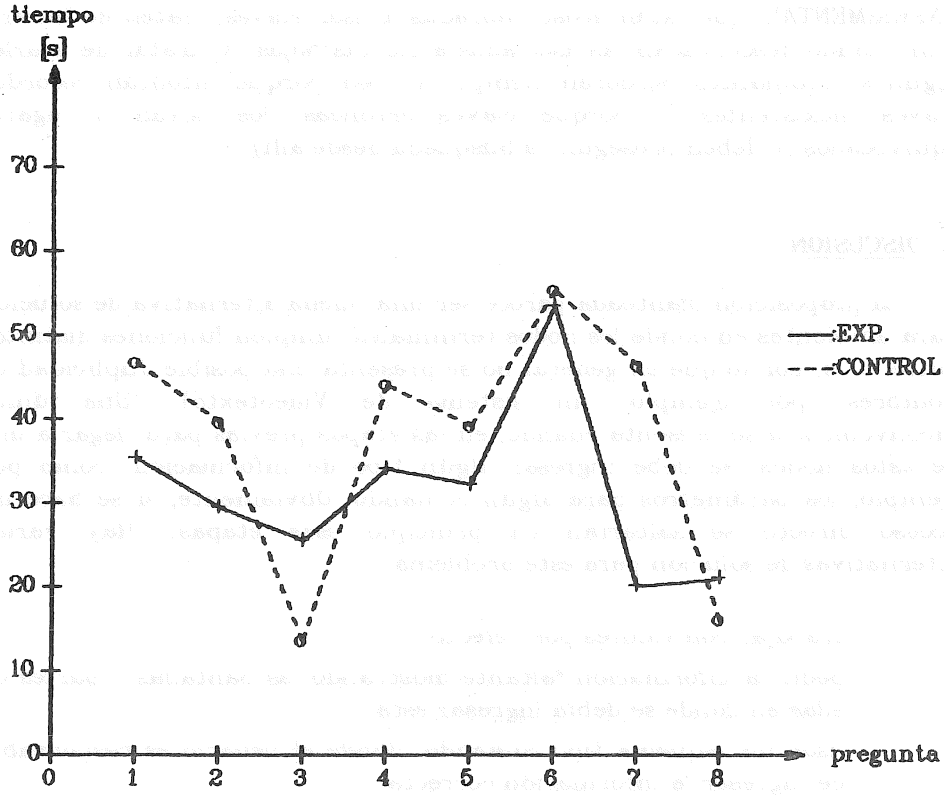


Figura No 1
Gráfico de Tiempos medios por grupo vs Preguntas

resto de la interfaz era idéntica para ambos grupos.

La gran diferencia en la pregunta 7 parece tener relación con su dificultad y con un aprendizaje en el uso de claves. La pregunta 4 fue similar a la 7: ambas requerían 8 pasos como mínimo para llegar a los nodos buscados, a partir de la pregunta anterior. Sin embargo, el desempeño de la pregunta 7 es mucho mejor que en la 4.

En las preguntas 3, 6 y 8, en principio el comportamiento debería haber sido muy similar, puesto que en general los sujetos del grupo EXPERIMENTAL no conocían las claves para esas hojas, y por lo tanto contaban con exactamente los mismo medios para llegar ahí que los del grupo de CONTROL. Sin embargo esto sucede sólo en la pregunta 6. En las preguntas 3 y 8, aúnque la diferencia no es significativa, hay una diferencia con signo positivo, lo que implica un mejor desempeño del grupo de CONTROL. Esto puede deberse a que las personas del grupo EXPERIMENTAL, que están acostumbradas a usar claves, traten de hacerlo aún cuando desconozcan las asociadas a ciertas hojas. Al tratar de usarlas algunos estudiantes perderán tiempo, ya sea porque intentan recordar claves inexistentes, o porque claves erróneas los llevan a lugares equivocados (y deben proseguir la búsqueda desde allí).

5. DISCUSION

La proposición planteada parece ser una buena alternativa de solución para ambientes en donde los nodos terminales cumplen funciones distintas entre ellos, por lo que en general no se presenta una posible duplicidad de nombres (por ejemplo, un sistema de Videotexto). Una única inconveniencia se presenta cuando, en las etapas previas para llegar a uno de estos nodos, se debe ingresar algún tipo de información, como por ejemplo, los parámetros para algún comando. Obviamente, si se hace un acceso directo se saltarían en principio esas etapas. Hay varias alternativas de solución para este problema:

- trabajar con valores por defecto.
- pedir la información faltante mostrando las pantallas o partes de ellas en donde se debía ingresar ésta.
- usar un esquema tipo comando, donde el usuario es responsable de ingresar la información correcta.

La idea propuesta puede implicar asignar nombres a una gran cantidad de nodos, dependiendo del tamaño del árbol de decisiones. En algunos casos esta alternativa puede ser improcedente por motivos tales

como ambigüuedades de funcionamiento o servicio de algunos nodos (por ejemplo, procesos de actualización, pero realizados sobre diferentes entes). En estos casos se deben buscar soluciones alternativas.

Por ejemplo, para el caso de los usuarios regulares, una forma de aminorar el tiempo perdido en el recorrido de la interfaz, es lograr que esta se adapte a sus necesidades. En particular lograr que aquellas que son más usadas se encuentren lo más cercanas posibles al primer nivel de la interfaz [5]. Esto implicaría decidir cuales son las más usadas, midiendo por ejemplo la frecuencia de uso de los nodos terminales, cuantificando luego el costo de acceso a ellos, y finalmente moviendo aquellos que sean muy usados y ésten lejanos del primer nivel. Dado que esto podría implicar tener muchas opciones en los niveles superiores, se deberían relegar a niveles inferiores aquellas opciones que son menos usadas.

Agradecimientos:
Los autores desean agredecer al Profesor Guido del Pino M. su apoyo estadístico.

6. REFERENCIAS

[1] Apperley M., Spence R.; "Hierarchical Dialogue Structure in Interactive Computer Systems", Software Practice and Experience, vol 13, no. 9, 1983, pp 777-790.

[2] Carroll, J.; "The adventure of getting to know a Computer", Computer, vol 15, no. 11, 1982, pp. 49-58.

[3] Dzida W., Herdza S., Itzfeldt; "User perceived quality of Interactive Systems", IEEE Transactions on Software Engineering, vol SE-4, no. 4, 1978, pp. 270-276.

[4] Gordon H., Stephen R.; "The Use of Menus in the Design of On-line Systems: A Retrospective View", SIGCHI, vol 7, no. 1,1985, pp. 16-22.

[5] Jofré M.; "Mejoras a Interfaces tipo menú", Tesis para optar al grado de Magister en Ciencias, mención Computación; Universidad de Chile, 1986.

[6] Jofré M., Pino J.; "Choosing Codes to Improve Menu Interfaces of Software Systems"; enviado para consideración a ACM SIGCHI Conference 1987.

[7] Morrison D.; Multivariate Statistical Methods; Mc Graw-Hill Inc., Nueva York, 1976.

Anexo

Cuestionario.

El cuestionario usado en estas pruebas contiene preguntas del tipo siguiente:

- Busque Economía de Recursos Naturales.
  Cuántos proyectos se realizan en esta área?

- Busque Departamento de Ingeniería Industrial.
  Pertenece el Sr. Yadlin a este departamento?

En el cuestionario usado en estas pruebas se incluyeron ocho preguntas. Para contestar estas preguntas, la persona debía tener acceso a alguna hoja específica del árbol de decisiones (ver Figura 2). En particular, cada pregunta requería visitar la hoja que se menciona a continuación (la numeración corresponde a la pregunta):

1- Ingeniería Forestal.
2- Economía de Recursos Naturales.
3- Economía Industrial
4- Departamento de Ingeniería Industrial
5- Ingeniería Forestal
6- Matemáticas del Uso de Recursos
7- Departamento de Ingeniería Industrial
8- Facultad de Agronomía

La Figura 2 incluye parte del árbol de decisiones del sistema usado en las pruebas.

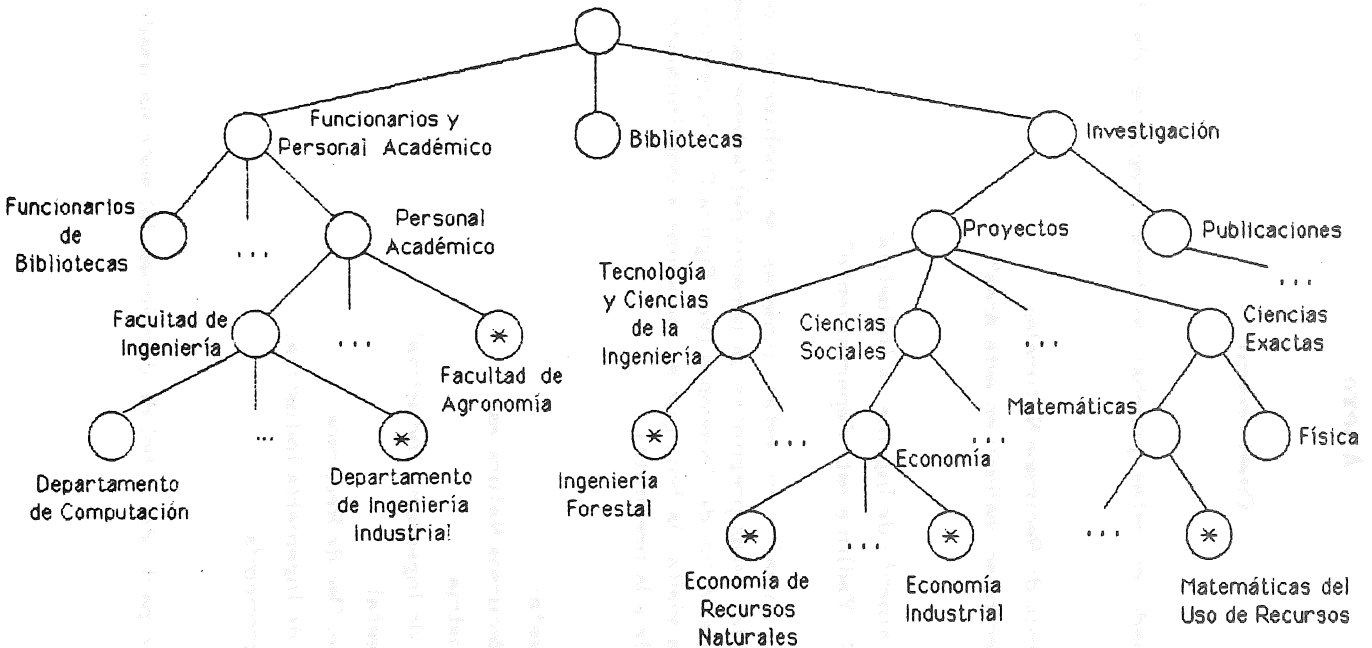


Figura No 2: Diagrama que muestra parte del árbol de decisiones usado.

(*): nodos visitados durante el recorrido realizado en el test de evaluación.

A CONCURRENCY CONTROL MECHANISM WHICH USES SEMANTIC KNOWLEDGE ABOUT THE APPLICATIONS

Amauri Marques Da Cunha
Universidade Federal do Rio de Janeiro
Rio de Janeiro - *Brasil*

1 - INTRODUCTION

The very fundamental idea which is the kernel of the flexible mechanism described in this work, is that the use of the semantic knowledge about the defined operations on the system objects, may lead to a greater parallelism in the entire system. This approach was studied from the theoretical point of view in [KUNG & PAPADIMITRIOU 79]. In this work, the authors demonstrated that a Concurrency Control mechanism will allow more parallelism, the more it knows about the semantics of the application.

For example, when the mechanism has knowledge of the objects' meaning and the defined operations on these objects, it may generate a greater number of correct **execution histories** [KUNG & PAPADIMITRIOU 79], then theoretically it will permit more concurrency in the system. The concurrency increased in this way, will be called **potential concurrency**, because we cannot guarantee that it will be advantageous in all real situations, without doing other practical considerations like overheads for example.

The issue of utilizing knowledge of the application to enhance Concurrency Control performance, was studied in several works. The most interesting propositions are [LAMPORT 76], [GARCIA-MOLINA 83], [LYNCH 83], [SCHWARZ & SPECTOR 84], [ALLCHIN & McKENDRY 83] and [WEIHL 83]. With regard to other works, the originality of the one presented here, is the proposition of a classification in **semantic knowledge levels**, of all the operations carried out on the system objects. This classification is defined in a naturally hierarchical way which allows, besides the specification of a Concurrency Control mechanism for each level of Semantic Knowledge, the grouping of these specifications in only one flexible mechanism. This mechanism is able to simultaneously manage the requests from all of the semantic levels. When we only have operations with a high degree of Semantic

Knowledge (SK), the mechanism will allow a high degree of potential concurrency. The potential concurrency increases dynamically according to the SK-level of the operations under execution on an object, if we use the proposed methodology and mechanism.

This paper is organized as follows. Section 2 presents the assumptions about the Computer System, and section 3 defines the SK-levels of the operations on the objects. Section 4 describes the necessary mechanisms for each level, and the 5th section is devoted to the important topic of recovery. Section 6 introduces the Concurrency Control Flexible Mechanism with SK utilization and gives an example. Section 7 presents a discussion about our approaches. Finally, in the 8th section we give our conclusion.

2 - ASSUMPTIONS ABOUT THE SYSTEM

The transaction concept has always been successfully used in the Data Bases domain. This concept has been studied and generalized during recent years -see |GRAY 81|. A good example of an extension to this concept, is the one of nested transactions |MOSS 81| and |BEERI et al. 83|.

More recently, several extensions have been proposed for the use of the transaction concept in the construction of distributed operating systems, which are not exclusively conceived for the Data Bases environment, but for more general applications. The reader will find in |SPECTOR & SCHWARZ 83|, a discussion on the subject. The new propositions that appear to be more interesting, are the following:

- The ARGUS Project at MIT |LISKOV 82| and |WEIHL & LISKOV 85|;
- The operating system CLOUDS at Georgia Institute of Technology |ALLCHIN & McKENDRY 83|;
- The ARCHONS Project at Carnegie-Mellon University |SHA et al. 83a| and |SHA et al. 83b|;
- The TABS prototype at Carnegie-Mellon too |SCHWARZ & SPECTOR 84| and |SPECTOR et al. 84|.

With the exception of SHA's works at Carnegie-Mellon, all of these propositions, explicitly modelize the object as **abstract types**. This modelization allows the inclusion of a greater variety of operations in the transactions, which represents a powerful extension for the manipulation of shared objects other than Data Bases.

This approach was adopted in the conception of the mechanism described in this work. We suppose that every system object is an instance of an **abstract data**

type, whose complete and formal specification is produced at the system design time |LISKOV & ZILLES 74|.

All the possible operations on a given object are rigorously defined, and therefore all of the operations' properties will be well known by the transactions that utilize the object. Each object is encapsulated by a kind of monitor that ensures its specification is respected. Additionally, we suppose that each site has only one object (this assumption is made only to simplify the exposition), and that each site has a kernel of the distributed operating system playing the role of transaction manager, as well as treating the synchronization problems, recovery, deadlock, and intersite communications. Other precisions about this approach may be obtained in |SCHWARZ & SPECTOR 84| and |WEIHL & LISKOV 85|.

This work deals particularly with the problem of transactions synchronization on objects. Its goal is to allow the creation of the greatest possible number of transactions **execution histories** on an object, without violation of the consistency constraints. For the intersite synchronization, one will use other techniques, for example the global serialization using timestamping ordering. We will not study these techniques here. For discussion of this subject, the reader could see for example |BERNSTEIN & GOODMAN 82|.

Finally, we chose the **two-phase locking** technique of |ESWARAN et al. 76|, as the basic technique for the proposed mechanism. If necessary, it is possible to combine this basic technique with a multitude of possible variations like the utilization of multiple versions, timestamping, etc. Even the adaptation to utilize the optimistic approach - which is radically opposed to the pessimistic two-phase locking - is easily feasible. Therefore, the mechanism presentation from this point of view, does not mean any loss of generality.

The deadlocks resolution, that is indispensable when one uses locks, is not explicitly treated here. Nevertheless, we will suppose the mechanism has the capability to accept requests to abort transactions, which might be generated by deadlock resolution procedures.

3 - SEMANTIC KNOWLEDGE LEVELS OF THE OBJECT OPERATIONS

This SK-levels classification, was mainly conceived to show the feasibility of a mechanism which is sensitive to the SK-level of the requests of operations made by transactions. A similar classification was proposed and studied from the theoretical point of view by |KUNG & PAPADIMITRIOU 79|.

Nevertheless, observing the more recent propositions of SK utilization to optimize the Concurrency Control, principally those which use the **abstract data types** approach such as |SCHWARZ & SPECTOR 84|, |WEIHL & LISKOV 85|, |WEIHL 83| and |ALLCHIN & McKENDRY 83|, we can remark that all of them have a tendency to mix the Concurrency Control mechanism with the object encapsulation. We mean by this that there is no clear separation between the Concurrency Control algorithms, and the object encapsulating procedures that implement its operations. In this model, the Concurrency Control mechanism has the right to observe the **object contents**, i.e. its internal structure and even the values therein stored, to decide about the compatibility of any two operations. This model was explicitly employed in |WEIHL 83|.

The work |SCHWARZ & SPECTOR 84| intented to separate the two functions, using the compatibility matrixes between the operations. Nevertheless, these compatibility matrixes of the examples given in |SCHWARZ & SPECTOR 84|, do not avoid the obligation that the Concurrency Control has to access the object content (value). So, this model is fundamentally the same as |WEIHL 83|.

To define the SK-levels, we initially established the following two categories according to whether the Concurrency Control mechanism does or does not have the right to observe the object contents to make its decisions. If it has this right, or more particularly if the requests of operation on the object, convey semantic information that allows the mechanism to efficiently use the object contents to decide about their compatibility, we can say that these requests are of a higher semantic level than the others. We consider that these requests are classified at the SK-level-three.

We chose to create a SK frontier at this point, because we consider that any mechanism which is capable of dealing with the SK-level-three will be more

resource consuming than the others which deal with the inferior levels. It should be used for an application, only if it has an advantageous price/performance ratio.

The requests that are of a level inferior to the SK-level-three, convey less semantic information. We consider the SK-level-one is attached to the fundamental operations defined on the object. We state further the SK-level-two, for the operation requests that convey more information than the fundamental operations of the SK-level-one. The additional information is represented by parameters added to the fundamental operations. These parameters amplify the quantity of available information to the Concurrency Control mechanism, leading to more parallelism in the system. The SK-level-zero and four are also defined, simply for completeness. They have already been identified in |KUNG & PAPADIMITRIOU 79| as the higher and lower possible levels of semantic knowledge respectively. In the following we present a SK-levels classification, illustrated by several examples, representing a pragmatic tentative in detailing the one already stated in |KUNG & PAPADIMITRIOU 79|:

SK-LEVEL-ZERO - the request of operation on the object, does not convey any Semantic Knowledge.

> **Example 0.1** - a request of an exclusive lock on the object, like for instance the classical lock WRITE (W) from the Data Bases domain.
>
> **Example 0.2** - in a Real Time System, after the occurrence of an alarm, an emergency transaction is launched. Before evaluating the extension of the problem indicated by this alarm, the transaction has to lock in an exclusive mode a certain number of objects.

SK-LEVEL-ONE - the request of operation on the object, is only composed by one of the fundamental operations that were defined at conception time of the **abstract type**. We recall that the object is an instance of such an **abstract type**. From this formal definition of the operations, one extracts a **compatibility table** between the operations, which gives all the ordered pairs of commutatives operations.

Example 1.1 - the operations READ (R) and WRITE (W) from the Data Bases.

representation of a compatibility table:

column: operation that owns a lock

line: operation that requests a lock

	R	W
R	Yes	No
W	No	No

Example 1.2 - suppose we have a **"strictly FIFO queue"**, with two types of operations defined on: INSERT (I) or DELETE (D) an element. By its own definition of a queue strictly FIFO, this queue does not admit the commutativity between two type I operations. The insertions of two elements have to be executed and validated, in the same order they arrived. The same applies for the type D operations. So, two operations of the same type cannot be compatible.

Furthermore, if the Concurrency Control mechanism only knows the requests arriving to it - in the present case I or D - and if it does not have the right to observe the interior of the object, then it cannot allow simultaneous executions of the I and D operations on the same object. In this situation it would not be capable of ensuring consistency. To demonstrate this, it is sufficient to suppose an initially empty queue. When two operations, one I and the other D, arrive, we can see that the execution order of these two operations is extremely important, because each different execution order gives a different result as well. Then, in this case, we cannot guarantee the commutativity between I and D. For this reason, its compatibility table will not allow any parallelism:

	I	D
I	No	No
D	No	No

Example 1.3 - suppose we have a **"weak FIFO queue"**, with the same operations as in example 1.2. As opposed to the strictly FIFO queue, the **weak FIFO queue** is defined as the one in which the Insert operations can be validated in an order different to the arriving order. The same thing applies to the Delete operations. It means that two type I operations are always commutatives, as are two type D operations. Otherwise, two operations of different types remain incompatible by the reasoning of the example 1.2. This gives the following compatibility table:

	I	D
I	Yes	No
D	No	Yes

This **"weak FIFO queue"** was adapted from |SCHWARZ & SPECTOR 84|, and it was also used in |WEIHL & LISKOV 85| with the name of **"semi-queue"**.

SK-LEVEL-TWO - at this level, we have all the level-one knowledge and, further, for each incompatibility, the possible existing commutativities according to a parameter added to the fundamental operation. This parameter does not change the general nature of the operation, however it conveys more detailed information about the operation meaning, creating in fact one sub-operation for each possible parameter. The parameter absence in a request of operation on an object, simply means that the request is SK-level-one.

Example 2.1 - a bank account object, on which are defined the operations READ (R) and WRITE (W) of the example 1.1, at the SK-level-one. Among all the semantic possibilities of the W operation, we suppose that the more frequent are the **deposits (d)** of any amount of money and the **withdrawals (w)** that do not exceed a certain threshold, as for instance the ones coming from an ATM - Automatic Teller Machine. This kind of withdrawal, which is already subject to constraints, is always paid by the bank. Then we will have the same compatibility table as in example 1.1 for the SK-level-one. When the request specifies a sub-operation W(d) or W(w), it will be necessary to see the following SK-level-two compatibility table:

	W(d)	W(w)
W(d)	Yes	Yes
W(w)	Yes	Yes

We observe in this example, that the perfect commutativities between the (sub-)operations d and w, lead to an important increase of the parallelism. In such a system, it is worthwhile specifying a second SK-level for the WRITE operations.

Example 2.2 - we retake the previous example 2.1. Now we suppose that the occurrence of simultaneous (sub-)operations W(w) will be a rare event in the system. So, the prohibition of the W(w)/W(w) commutativity will penalize only slightly the performance of the new system, with regard to that of example 2.1. In other respects, we estimate that the serialization of all W(w) (sub-)operations, that are made imperative if all pairs W(w)/W(w) become incompatible, will greatly facilitate the design of the other system modules. As an example we can mention protection against errors and frauds. Hence, such a system simplification would bring about a decrease in the number of routines, as well as a reduction in their complexity, leading to an improvement of the overall performance. Here is an illustration of utilization of SK-levels, that may give more flexibility to the System design, in addition to the potential gains of parallelism. In this case, we use the compatibility table below, instead of the table in example 2.1.

	W(d)	W(w)
W(d)	Yes	Yes
W(w)	Yes	No

Example 2.3 - suppose we have a directory object, where each element (entry) is identified by a key formed by a character string. Its fundamental operations are MODIFY (M) an element - which may be insert or delete an element, or simply modify its contents - or LOOKUP (L) an element, or still DUMP (D) the entire directory. At the SK-level-one, the M and L operations cannot indicate the key of the

target element. The D operation is only defined at this level. Here is the compatibility table of the SK-level-one:

	M	L	D
M	N(*)	N(*)	N
L	N(*)	Yes	Yes
D	N	Yes	Yes

(*) the incompatibilities marked with an asterisk, may direct the decision to the SK-level-two, provided that the key of the target element is present as an operation parameter.

Nevertheless, we will not have here a compatibility table to characterize the SK-level-two. The relations M==>M (here the symbol ==> means precede), L==>M and M==>L do not create dependencies [SCHWARZ & SPECTOR 84], when each operation is realized on a different element of the directory. Therefore it is sufficient to keep a list that contains all the parameters of the M or L operations which own a lock on the object. The verification of the commutativity of one (sub-)operation M(k) or L(k) just arrived (k = key of the directory element addressed by the operation), with the ones already in execution, is done by a simple search on the list (table) of accepted locks.

We recall that the commutativity L(k)/L(k'), even if k'=k, is detected at the SK-level-one, so the test does not have to be done at the SK-level-two.

SK-LEVEL-THREE - at this level, we have the sum of the level-one and level-two knowledge and, in addition, we know that the operations commutativity depends on the object value.

Example 3.1 - a bank account object that has, among the operations defined on it, a general operation of money withdrawal that brings together all the kinds of possible withdrawals. In this case, two withdrawal operations are commutative only if there is enough money in the account.

Example 3.2 - we retake the "strictly FIFO queue" of example 1.3. The incompatibilities I/D and D/I may be removed by a procedure of the Concurrency Control mechanism, that is capable of observing the object contents to make its decision in the following manner:

From example 1.2, we already know that when the queue is empty, there is no commutativity between I and D. There are other equivalent situations, and to show it let us state the variables:

n_q = number of elements (validated) of the queue

n_D = number of D operations that own a lock on the object

n_I = number of I operations that own a lock on the object

Effectively, the limit-situation is reached when $n_q=n_D$, i.e. at the moment the queue is potentially empty. In reality, the queue only becomes truly empty, if all of the D-operations that own a lock on the queue, are validated (we cannot forget that aborts are still possible before validation). Anyway, since we have $n_q<n_D$, we cannot guarantee that a just arrived I-operation, will be compatible with all the E-operations that already possess the lock. To demonstrate this, we suppose that this lock I is granted. If this operation I is validated before the validation of the last D-operation, we will have a different result from the one that would be obtained if the validation of I had been done after the validation of the last E. In this case, the commutativity hypothesis is contradicted, and so the assertion is demonstrated.

It is interesting to observe that, even if the different results mentioned in the above demonstration, do not seem to cause consistency problems, they have to be strictly prohibited. Why?!!... Because the Concurrency Control mechanism has to guarantee a global serialization of the transactions, to ensure the system's global consistency. The possible local commutativities, that allow more parallelism and hence more potential concurrency, are admitted on the condition that it permits (or does not prevent) the construction of a global execution order of the transactions [GARDARIN & MELKANOFF 82]. Thus, the operations on an object that have the risk of giving different results from the semantic point of view, according to their execution (validation) ordering, have all to be serialized at the local level.

Returning to the example, if we have a D-operation arriving and other I-operations that already own a lock, we consider that the limit-situation still depends on the variables n_q and n_D. Effectively, the worst case happens when all the D-operations are validated before all the I-operations. Thus to grant a D-lock on the object concerned, supposing that it already has other I-locks granted, it is necessary that the condition $n_q > n_D$ be true. Furthermore, if we have a request for an I-lock, and the object has already granted other D-locks, we will grant the lock only in the case of $n_q >= n_D$ (we notice that the equality is admitted here).

Example 3.3 - we take again the **"weak FIFO queue"** of example 1.3. Following the reasoning used in example 3.2, we can deduce that even the compatibility D/D, which seemed to be completely natural, could only be decided at the SK-level-three. If we suppose there are only D-locks granted, an arriving request for a new D-lock, can only be accepted if $n_q > n_D$ or if n_q=zero. We summarize the Concurrency Control rules on this object as follows:

a - compatibility table SK-level-one:

	I	D
I	Yes	No
D	No	No

b - SK-level-three procedures:

b.1 - incompatibility of SK-level-one I/D: the request for the I-lock, is granted only if $n_q >= n_D$;

b.2 - D/I: the request D is granted if $n_q > n_D$;

b.3 - D/D: the request is granted if $n_q > n_D$ or if n_q=zero.

Observation - in the given examples, we only studied one exceptional situation of the queues: the possibility that it becomes empty. Clearly, this is the most important exception. However, in practice, we could rarely suppose a queue to be infinite. In these cases, we would also have to treat the possibility of having a full queue. If this happens, we could use reasoning which is entirely analogous to the reasoning used in examples 3.2 and 3.3. For instance, the compatibility I/I will not be

possible at the SK-level-one; it will also be decided at the SK-level-three, using the maximum dimension of the queue.

SK-LEVEL-FOUR - we can imagine that it will be possible in the future, to store and/or analyse the complete Semantic Knowledge about all the transactions and all the objects within a distributed system. With this knowledge, the Concurrency Control mechanism could find all the possibilities of commutativity between operations, as soon as the operations arrive. This highest SK-level, corresponds to the **"optimal schedulers"** that utilize the maximum information possible about the transaction system [KUNG & PAPADIMITRIOU 79]. This type of scheduler has the capability to generate all the possible correct execution histories (logs), and therefore it is optimal with respect to the parallelism.

4 - THE NECESSARY MECHANISMS FOR EACH LEVEL

We need to make some comments about the SK-level classification, before the presentation of the mechanisms.

Initially we notice that there is no sense in classifying at the SK-level-one, a fundamental operation that needs an exclusive access to the object. From the point of view of the approach introduced here, this operation should in reality be located at the SK-level-zero. Even at the SK-level-two, we do not have to define a sub-operation that is incompatible with all the others at the same level. If this happens, we can say that in fact it can be reduced to a SK-level-one, or even to a different SK-level-zero operation.

The WRITE (W) and READ (R) operations, classic in the Concurrency Control study for the Data Bases, already represent a certain SK-level. With our approach, the W operation - that requires an exclusive access to the object - would be classified at the SK-level-zero. At the SK-level-one, we would only have the R operation, which is always compatible with itself.

Finally it is interesting to observe that the SK-level classification can be augmented by the definition of other intermediate levels. It is easy to conceive

for example that the level-two operations may still be divided in other sub-operations by the adjunction of other parameters. Each parameter added to a basic operation, may establish a new sub-level, with its corresponding compatibility tables. The classification proposed here, seems to be a general framework, and is able to accept some refinements concerning how to express the Semantic Knowledge degrees of transaction systems.

4.1 - The Concurrency Control mechanisms used at each SK-level

SK-LEVEL-ZERO - the guarantee of exclusive access, may be obtained by managing a semaphore for each object.

SK-LEVEL-ONE - this is a well-known case, and there is a vast literature about it in the Data Bases domain. From the definition of the fundamental operations, we establish a compatibility table between the operations. The utilization of this table with the two-phase locking protocol [ESWARAN et al. 76], ensures global consistency. Obviously the mechanism will also manipulate a table which keeps the granted locks.

SK-LEVEL-TWO - it is a generalization of the SK-level-one. According to the object semantics, we can distinguish two types of implementations.

In the first - as in example 2.1 of the previous section - we have one or several compatibility tables of the SK-level-two, each one corresponding to a pair of incompatible operations at the SK-level-one. A SK-level-two table, furnishes the information about the compatibility between the different existent sub-operations, derived from the fundamental operations. The Concurrency Control algorithm, in deciding about the compatibility between a certain pair of sub-operations, must first find the appropriate compatibility table, and then it applies the two-phase locking protocol as in the SK-level-one.

The second type of implementation is for instance, the one in example 2.3. Here, instead of consulting a compatibility table for the SK-level-two, the algorithm stores in a list, the parameter of each sub-operation that has a granted lock. This supplementary storage may be added to the proper lock table to facilitate the manipulations. An arriving lock request, will only be accepted if its parameter is not in the list (table) of granted locks. Naturally the logical

complement of this algorithm, should be available as a variation of this type of implementation; in this case it should only accept the sub-operations having the same parameter as another already in the granted locks table. Obviously, this second type of implementation fits only in the situations where the parameters' diversity cannot be limited, or is too great, as in example 2.3. Otherwise, the first type with its compatibility tables would be sufficient.

The choice of the type of implementation and its possible variations, has to be done as a preliminary, at the very moment of the conception and encapsulation of the abstract object.

SK-LEVEL-THREE - at this level, the mechanism has the right to observe the object contents to make its decisions. Example 3.1 of the previous section, illustrates this idea very well.

Each object that is able to accept SK-level-three operations, will be initialized with a special-purpose procedure, that verifies the commutativity of an arriving request of operation on the object, with all the others which already own a lock at that moment. This procedure may go as far as consultating the semantic information conveyed by all the other operations that are present in the lock table. This procedure may even execute the just arrived operation, using the current (validated) object value, to accomplish its verification.

However, there are simpler situations, as in examples 3.2 and 3.3. Effectively, the necessary information on these objects, can be summarized within counters for these examples. If these counters are stored in main memory, this access to the object will not be much more expensive than the (mandatory) access to the lock table.

SK-LEVEL-FOUR - we are not yet capable of presenting the specifications for this level. Nevertheless, previous work [LAMPORT 76] has demonstrated its feasibility.

5 - THE RECOVERY PROBLEM

The need to abort a transaction, may appear at random. The main events that give rise to an abort are:

- faults
- deadlocks
- Promptness Control
- user request

Apart from in very special-purpose systems, normally the Concurrency Control mechanism has to be able to deal with aborts in a way that preserves consistency, and has to allow transactions recovery. The classical way to abort a transaction, is to undo it, by resetting all the objects that were modified by the transaction, to the initial state encountered by the transaction. However, when we use semantic knowledge that induce the commutativity of operations that **modify the object state**, we cannot simply reset the initial state, because this would lead the system to an inconsistent state. We recall that the commutativity of the operations on an object, allows that the local order of execution (and validation) of the commutative operations, is different from the global serialization order of the transactions which those operations belong to.

Thus, it is necessary to have a supplementary hypothesis to solve this problem. We make the same supposition as that proposed by [GARCIA-MOLINA 83], i.e., we suppose that each operation involved in a commutativity relation, has a counter-operation that is capable of undoing from the semantic point of view the original operation. This counter-operation has to be imperatively commutative with the other operations and counter-operations belonging to the same commutative set of its original operation.

A counter-operation may be represented either by the inverse operation on the abstract object, or by another compensating operation. In any case it must not demand the allocation of objects other than the object already concerned, because on this object, the transaction has always the suitable lock, even during the abort execution.

This additional hypothesis is, of course, very strong and, consequently, greatly reduces the applicability of the proposed semantic levels. We note that even the

existence of commutative operations, was an already very restrictive characteristic, to which we are obliged to add the existence of the corresponding counter-operations. On the other hand, there are certainly many particular systems, where the mechanisms proposed here, considerably increase the concurency all over the system - see for instance |GARCIA-MOLINA 83|, |LYNCH 83|, |SCHWARZ & SPECTOR 84|, |WEIHL & LISKOV 85| and |WEIHL 83|.

6 - THE CONCURRENCY CONTROL FLEXIBLE MECHANISM

Observing the description of the Semantic Knowledge levels zero, one, two or four, one notices that there is a perfect hierarchy between these levels. The SK-level-zero does not permit any parallelism on the object. The SK-level-one has a compatibility table that establishes the possible permitted concurrencies between the operations defined on the object. The SK-level-two only exists for each pair of SK-level-one incompatible operations. Hence, we clearly see the possibility of simultaneously controlling the two levels: the one and the two. It is sufficient to have a hierarchical structure on which the locking is made, and that this structure be attached to the existent compatibility tables within each level. This reasoning can equally be applied to the SK-level-three, that can be reached either bv a SK-level-two incompatibility or by another SK-level-one incompatibility. To correctly solve all concurrency cases that may occur on the object, it will be sufficient to control an unique tree of locks.

Thus, we can use an unique mechanism to control the concurrency on a object, disregarding the SK-level of the transactions that utilize this object. And further, such a flexible mechanism, will allow more parallelism as the transactions (operations) Semantic Knowledge is higher. It dynamically adapts itself, in a natural way with respect to the arriving transactions (operations), i.e., if it treats a group of requests that conveys a high SK-level, it will allow more potential parallelism. On the other hand, when an arriving group of requests (of operations) has at least one low semantic level request, the degree of concurrency automatically decreases by the mechanism functioning during all the period this request owns its lock.

6.1 - An example of the Flexible Mechanism

We suppose an object that accepts the SK-levels one and two. The fundamental operations defined on it, are the current ones usually made in the Data Bases Systems domain:

R - only has the right to read the object

W - has the right to modify the object

The SK-level-one is described by the following compatibility matrix CM1:

CM1:

	R	W
R	1	0
W	0	0

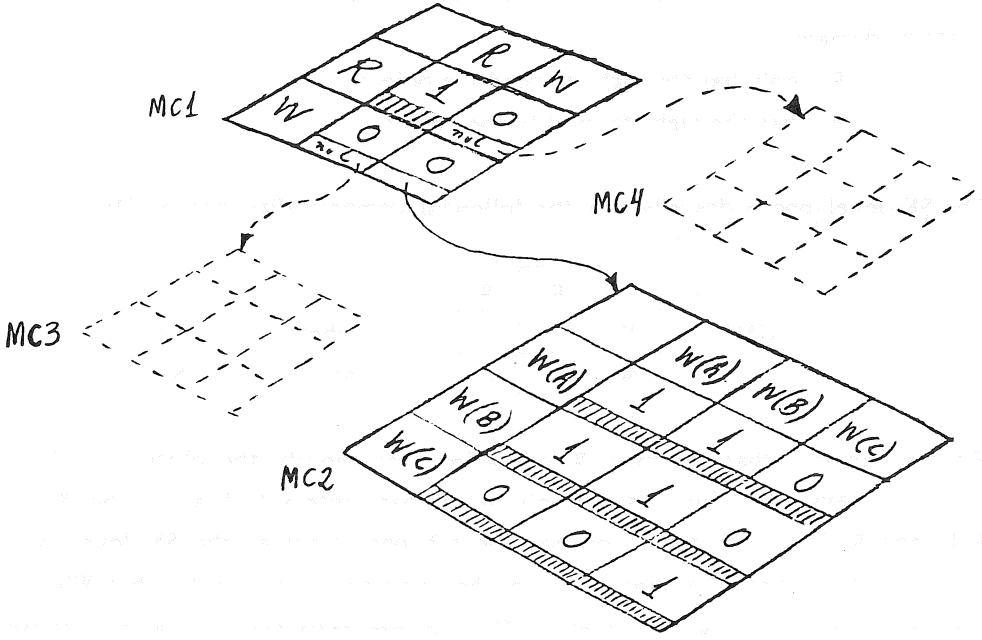
where 1=Yes
and 0=No

We also suppose that the type W operations can modify the object, in three different ways, identified respectively by the parameters A, B and C. So, W(A), W(B) and W(C) will be the three possible sub-operations at the SK-level-two. We still admit that W(A) and W(B) can be executed in any order, but W(C) is commutative only with itself. The commutativities between these sub-operations, are expressed by the compatibility matrix CM2:

CM2:

	W(A)	W(B)	W(C)
W(A)	1	1	0
W(B)	1	1	0
W(C)	0	0	1

To show clearly the hierarchical aspect, we represent the matrixes together, adding a pointer (that may contain the nil value) to the SK-level-two within each CM1 element. The following figure represents this:


MC1
R
R
W
R
1
W
W
0
0
MC4
MC3
MC2
W(A)
W(B)
W(C)
W(A)
1
W(B)
1
W(B)
W(C)
W(C)
1
1
0
0
0
1

If the incompatibilities R/W and W/R had some possibilities of compatibility at the SK-level-two, they could be represented for example, by the matrixes CM3 and CM4 of the above figure.

The operations requested by the transactions, always have a well specified SK-level. In our example, if a W operation does not convey a parameter, it will be considered a SK-level-one request. During all the time this operation has a lock, it has to prohibit the simultaneous execution of other W's, even if they are of the SK-level-two. Therefore, a SK-level-one lock, can block one or several possibilities of the SK-level-two - each set of (semantic) possibilities being represented by a SK-level-two compatibility matrix.

On the other hand, each SK-level-two W operation will only allow (possibly) other concurrent W's of the same SK-level. As long as there is at least one SK-level-two W, no other SK-level-one W (and no other R either) can be executed.

This example shows the need of a hierarchical locking table for this kind of mechanism. The annex shows an implementation of the proposed mechanism, and also describes its more important aspects.

7 - A DISCUSSION ABOUT OUR APPROACHES

The two principal approaches used in the conception of the proposed mechanism, are discussed in this section.They are the shared abstract types approach to model the system's objects, and the semantic knowledge levels approach that utilizes semantic knowledge about the application.

7.1 - The Shared Abstract Types Approach

This approach was well studied in [SCHWARZ & SPECTOR 84]. The same approach has been developped at the MIT with the name of atomic data types [WEIHL & LISKOV 85].

7.1.1 - ADVANTAGES

The first and principal advantage is inherent to this approach. Effectively, the manner of modeling objects as abstract types instanciations, is already oriented towards the distributed model of computing, therefore facilitating the development of distributed systems in general. We recall that the object is the unit of permanent data within a system. This unit is used to control the concurrent transactions. On the other hand, the manner of modeling objects in the classical Data Bases approach, requires the utilization of special and often very expensive techniques to pass from the centralized to a distributed framework.

The following advantages, are listed according to the classification of required characteristics of a Real Time Distributed System, described in |LE LANN 83|.

FLEXIBILITY: this approach encapsulates the objects with some modules, which constrains the objects to behave according to their specifications. Hence, there is an obvious modularity in this approach. To change or adapt the behaviour of an object, to fit new specifications, we can envelop it with a new encapsulation module |WEIHL & LISKOV 85|. We can still encapsulate several different objects to unite them. This modularity considerably increases the system's flexibility.

CORRECTNESS: an abstract type specification may be done through a mathematical formalization. The proof of correctness of all the operations on an object, is somewhat inherent to this approach. Consequently, this rigor facilitates the transactions construction, and also the establishment of a formal demonstration of the application's correctness |LISKOV & ZILLES 74|.

RELIABILITY: the abstract type approach makes possible an atomic treatment of the object, as the one studied in |WEIHL & LISKOV 85|. Thus, adding recovery procedures and even fault tolerant procedures to the object encapsulation |WEIHL & LISKOV 85|, we could ensure a great reliability at the local level. Its extension to the global level will depend on other mechanisms, which will cope with the global atomicity and resilience, and will enhance the local reliability.

PROMPTNESS: in some particular cases, this characteristic may be improved. We take the example of an object that admits compensating operations, like the ones mentioned in section 5. We can imagine that it will be possible in this case, to anticipate the validation of an operation that modifies the object. If the possibility of compensation does not endanger the consistency, therefore we can obtain an improvement on the promptness.

7.1.2 - DISADVANTAGES

SPECIFICATION OF THE ABSTRACT TYPES: this is an inherent disadvantage of this approach. We are obliged to conceive and develop (and still program), the abstract types that are necessary to the application. This handicap is represented in practice by a high development cost in comparison with the other classical approaches. We already know that in the future, we will have languages and/or distributed operating systems, which will offer some primitives to assist the specification of abstract types, and even possibly, the complete encapsulation for the more frequently used types [WEIHL & LISKOV 85]. Though this disadvantage tends to decrease with time, it will not be eliminated.

OVERHEAD: obviously, this approach considerably increases the overhead for all the object utilizations. The first experiments reported, for example the TABS system from Carnegie-Mellon University [SPECTOR et al. 84], confirm this assertion. However, we are sure that at least for the small transactions or in general for applications that are simple, this approach is not worthwhile. On the other hand, for the distributed systems of great complexity, that have large size transactions, we can catch a glimpse of good results for the parallelism degree, for the management of the complexity, etc. It is still necessary to work out a lot of studies and researches to find the trade-offs.

7.2 - The Semantic Levels Approach

7.2.1 - ADVANTAGES

I - the first advantage is to allow much more potential concurrency in numerous cases, like for example the ones mentioned in section 3.

II - this approach only adds an overhead to the Concurrency Control mechanism, if it has to cope with the highest SK-levels. Indeed, take the example of an implementation of the SK flexible mechanism presented in section 6. Observing it, we ascertain its modularity with respect to the SK-levels. The semantic lock table, the storing of the compatibilities, and the procedures corresponding to the highest SK-levels (especially two and three), are only used to treat the operations requests with these highest levels. As soon as we are sure of having only SK low levels operations (as zero or one for example), we can remove practically all that were added to manage the other levels, bringing the mechanism to the same overhead of a classical mechanism.

This reasoning can also be applied to the dynamic behaviour of the system. Suppose we have on a site, the complete mechanism to cope with all the SK-levels. To manage the requests of operations of the SK-level-zero or SK-level-one, the mechanism will not have in practice any increase in its time overhead with regard to the classical mechanisms. Only in the case of the arrival of higher SK-level requests, will the mechanism consume more execution time. Nevertheless, at this moment, we will have other gains from the point of view of the parallelism.

III - the utilization of SK-levels, allows us to obtain some profits with a common situation in the practice of transaction systems. Particularly in the Real Time Systems, the transactions are specified, developed and catalogued previously, to be dynamically launched later during the system's life [DA CUNHA & TEIXEIRA 84]. In order to keep a sufficient degree of generality for the transactions, we can think they will be catalogued with the lowest possible SK-level about the objects. Otherwise, at the moment of a transaction dispatching, the transactions manager will have more precisions about the current state of the system,

allowing in this manner to augment the SK-level of the transaction. Thus, we can envisage the "refinement" of the SK of a transaction already catalogued, dynamically from the moment it has to be launched. This "refinement" may also occur at the moment of dispatching a local action on a site, because at this moment it could take advantage of the known results of the (already executed) previous actions.

The example 0.2 in section 3, illustrates this situation, with the emergency transactions of a Real Time System.

IV - the above advantage refers to the strategy of static allocation of resources. The strategy of dynamic allocation can however take advantage of SK-levels utilization. Here, the moment when the transaction requests the lock(s), i.e., the moment of dispatching the local action on a site, it is precisely when there is the highest possible SK, and then this is the best moment to decrease the probability of conflicts, and consequently increase the potential parallelism. In attaining a diminution of the probability of conflicts occurrence, one favours the dynamic allocation of objects by the decreasing of the aborts percentage, which is the principal handicap of this strategy.

7.2.2 - DISADVANTAGES

A major disadvantage of this approach, is the limitation of its utilization to only certain kind of applications. We refer the reader to section 5, for a discussion of this subject.

8 - CONCLUSION

We recall that the theoretical base of the proposed mechanism, is the work developed in [KUNG & PAPADIMITRIOU 79], where the authors demonstrated the existent relation between the Concurrency Control permissiveness and the semantic knowledge about the application. Using this idea, we defined four levels of Semantic Knowledge (SK) that are hierarchically organized. We suppose the objects are modeled as abstract types. Thus, each request of an operation on an object, conveys a well-known SK-level, which is used by our SK

Flexible Mechanism. The higher the SK-levels of the requests received by the mechanism on an object, the more parallelism it allows on this object. Using our Flexible Mechanism, the parallelism increases and decreases automatically, according to the SK-levels of the requests on the object.

In the literature, we find a similar idea in the protocol proposed by |GRAY et al. 76|, to deal with the granularity of objects in a Data Base. Our proposed framework can also be applied to control the concurrency on variable granularity objects - see example 2.3 of section 3. Nevertheless, the hierarchical protocol used in |GRAY et al. 76|, is based on the more "physical" concept of object granularity, whereas ours is based on the logical nature of the operations on the object. So, our proposition is applicable to more general situations - see section 3. We can still note that the **"intention modes"** of |GRAY et al. 76|, when applied to a node, prevent undesirable accesses to the underlying structure. There is an analogous situation in our mechanism, when a low SK-level operation acquires a lock on a object and therefore blocks undesirable accesses even if they have a high SK-level (see section 6).

The work |SCHWARZ & SPECTOR 84| had a lot of influence on our proposition. Our approaches are significantly the same. From the description of the examples studied in |SCHWARZ & SPECTOR 84|, it emerges that their Concurrency Control mechanism can explicitly utilize the parameters of the fundamental operations defined on the object, in a way that is entirely analogous to our SK-level-two. Furthermore, their mechanism needs in several cases, to observe the object contents, as in our SK-level-three. This appears only implicitly in the paper |SCHWARZ & SPECTOR 84|, where the mechanism is always presented in the form of a compatibility table. In this manner, they cannot treat correctly the **"weak FIFO queue"** of example 3.3 of section 3, in the cases where the queue may be contingently empty (or contingently full).

Hence, the utilization of compatibility tables between the operations - even adding parameters to these operations - does not seem sufficient to correctly express all the possible cases of concurrency on the abstract types. Effectively, it is still necessary to add special-purpose procedures, that are capable of "refining" the verification of the commutativity, in all the possible situations that are not provided by the compatibility table. This is done in a very natural way in our SK-levels proposition, but is not explicitly treated in the proposition

|SCHWARZ & SPECTOR 84|.

The paper |WEIHL 83| introduces the notion of **dynamic atomicity**. It models the Concurrency Control mechanism (scheduler) and the encapsulation of the object (modeled itself as an abstract type), into a single module. This **dynamic atomicity** clearly corresponds to our SK-level-three. The framework of |WEIHL 83| is less flexible than ours, because it does not allow simpler (and less expensive) treatments.

Finally, we comment on the propositions of the CLOUDS Operating System |ALLCHIN & McKENDRY 83| with respect to ours. CLOUDS proposed four conceptual levels for the Concurrency Control mechanism, with the same idea that the higher the level, the more semantic information is available to the mechanism, and so the more parallelism is allowed. There is a fundamental difference between the conceptual levels of CLOUDS and the SK-levels. The CLOUDS conceptual levels are mutually exclusive by definition, i.e., they cannot simultaneously coexist within the same object and in the same application. This is due to the fact that they were not defined in a hierarchical manner, and so the mechanism cannot traverse, in a way that preserves the consistency, from one level to another, as we can do with the SK-levels. In this sense, our proposition allows much more flexibility.

We hope the framework proposed here, may lead to a more comprehensive study of the Concurrency Control on objects modeled as abstract types.

REFERENCES

|ALLCHIN & McKENDRY 82| J.E. ALLCHIN et M.S. McKENDRY
"Object-based Synchronization and Recovery"
Georgia Institute of Technology, Technical Report GIT-ICS-82/15, (septembre-1982) 20p.

|ALLCHIN & McKENDRY 83| J.E. ALLCHIN et M.S. McKENDRY
"Synchronization and Recovery of Actions"
Proc. 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Montreal (août-1983) pp.31-34

|BEERI et al. 83| C. BEERI, P.A. BERNSTEIN, N. GOODMAN, M.Y. LAI et D.E. SHASHA
"A Concurrency Control Theory for Nested Transactions"
Proc. 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Montreal (août-1983) pp.45-62

|BERNSTEIN et al. 80| P.A. BERNSTEIN, D.W. SHIPMAN et J.B.ROTHNIE JR.
"Concurrency Control in a System for Distributed Databases (SDD-1)"
ACM Transactions on Database Systems, Vol.5, No.1 (mars-1980) pp.18-51

|BERNSTEIN & GOODMAN 81| P.A. BERNSTEIN et N. GOODMAN
"Concurrency Control in Distributed Database Systems"
Computing Surveys, Vol.13, No.2 (juin-1981) pp.185-221

|BERNSTEIN & GOODMAN 82| P.A. BERNSTEIN et N. GOODMAN
"A Sophisticate's Introduction to Distributed Database Concurrency Control", Proc. 8th Int. Conf. on Very Large Data Bases, Mexico (septembre-1982) pp.62-76

|CAREY & STONEBRAKER 84| M.J. CAREY et M.R. STONEBRAKER
"The Performance of Concurrency Control Algorithms for Database Management Systems", Proc. 10th Int. Conf. on Very Large Data Bases Singapore (août-1984) pp.107-118

|DA CUNHA & TEIXEIRA 84| A.M. DA CUNHA et M.J. TEIXEIRA
"SIGMA - Um Executivo Distribuido para Ambientes Automatizados em Tempo Real", Proc. 5° Congr. Bras. Automatica
Campina Grande (septembre-1984) pp.517-522

|ESWARAN et al. 76| K.P. ESWARAN, J.N. GRAY, R.A. LORIE et I.L. TRAIGER, "The Notions of Consistency and Predicate Locks in a Database System", Communications of the ACM, Vol.19 No.11 (novembre-1976) pp.624-633

|GARCIA-MOLINA 83| H. GARCIA-MOLINA "Using Semantic Knowledge for Transaction Processing in a Distributed Database", ACM Transactions on Database Systems, Vol.8 No.2 (juin-1983) pp.186-213

|GARDARIN & LEBEUX 77| G. GARDARIN et P. LEBEUX "Scheduling Algorithms for Avoiding Inconsistency in Large Databases" Proc. 3rd Int. Conf. on Very Large Data Bases, Tokyo (octobre-1977) pp.501-506

|GARDARIN & MELKANOFF 82| G. GARDARIN et M. MELKANOFF "Concurrency Control Principles in Distributed and Centralized Databases" INRIA - Rapport de Recherche No.113 (janvier-1982) 91p.

|GRAY et al. 76| J.N. GRAY, R.A. LORIE, G.R. PUTZOLU et I.L. TRAIGER "Granularity of Locks and Degrees of Consistency in a Shared Data Base" Proc. IFIP Working Conf. on Modelling in Data Base Management Systems Freudenstad (janvier-1976) pp.695-723

|GRAY 81| J.N. GRAY "The Transaction Concept: Virtues and Limitations", Proc. 7th Int. Conf. on Very Large Data Bases, Cannes (septembre-1981) pp.144-154

|KORTH 83| H.F. KORTH "Locking Primitives in a Database System", Journal of the ACM, Vol.30 No.1 (janvier-1983) pp.55-79

|KUNG & PAPADIMITRIOU 79| H.T. KUNG et C.H. PAPADIMITRIOU "An Optimality Theory of Concurrency Control for Databases" Proc. ACM SIGMOD Int. Conf. on Management of Data, Boston (mai-1979) pp.116-126

|LAMPORT 76| L. LAMPORT
"Towards a Theory of Correctness for Multi-user Data Base Systems"
Technical Report CA-7610-0712, Massachusetts Computer Associates Inc. (octobre-1976)

|LE LANN 83| G. LE LANN
"On Real-Time Distributed Computing"
Proc. IFIP-83 Congress, Paris (septembre-1983) pp.741-753

|LISKOV 82| B. LISKOV
"On Linguistic Support for Distributed Programs"
IEEE Transactions on Software Engineering, Vol.SE-8 No.3 (mai-1982) pp.203-210

|LISKOV & ZILLES 74| B. LISKOV et S. ZILLES
"Programming with Abstract Data Types"
Proc. ACM SIGPLAN Symposium on Very High Level Languages
Santa Monica (mars-1974) pp.50-59

|LYNCH 83| N.A. LYNCH
"Multilevel Atomicity - A New Correctness Criterion for Database Concurrency Control"
ACM Transactions on Database Systems, Vol.8, No.4 (décembre-1983) pp.484-502

|MOSS 81| J.E.B. MOSS
"Nested Transactions: An Approach to Reliable Distributed Computing"
Thèse Ph.D. MIT, Technical Report MIT/LCS/TR-260 (avril-1981) 178p.

|SCHWARZ & SPECTOR 84| P.M. SCHWARZ et A.Z. SPECTOR
"Synchronizing Shared Abstract Types"
ACM Transactions on Computer Systems, Vol.2, No.3 (août-1984) pp.223-250

|SHA et al. 83a| L. SHA, J.P. LEHOCZKY, E.D. JENSEN et N. PLESZKOCH
"Data Consistency and Transaction Correctness - A Modular Approach to Non-serializable Transactions"
CARNEGIE-MELLON University, draft (16-août-1983) 28p.

[SHA et al. 83b] L. SHA, E.D. JENSEN, R.F. RASHID et J.D. NORTHCUTT
"Distributed Co-operating Process and Transactions"
in Distributed Computing Systems - Synchronization, Control and Communication, Y.PAKER ed., ACADEMIC PRESS (1983) pp.23-50

[SHA 84] L. SHA
"Modular Concurrency Control and Failure Recovery ...Consistency, Correctness and Optimality"
Thèse en préparation à l'Université CARNEGIE-MELLON,
draft (29-août-1984) 12p.

[SPECTOR & SCHWARZ 83] A.Z. SPECTOR et P.M. SCHWARZ
"Transactions: A Construct for Reliable Distributed Computing"
ACM Operating Systems Review, Vol.17, No.2 (avril-1983) pp.18-35

[SPECTOR et al. 84] A.Z. SPECTOR, J. BUTCHER, D.S. DANIELS, D.J. DUCHAMP, J.L. EPPINGER, C.E. FINEMAN, A. HEDDAYA & P.M. SCHWARZ
"Support for Distributed Transactions in the TABS Prototype"
CARNEGIE-MELLON University,
Technical Report CMU-CS-84-132 (juillet-1984) 25p.

[WEIHL 83] W.E. WEIHL
"Data-dependent Concurrency Control and Recovery"
Proc. 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Montreal (août-1983) pp.63-75

[WEIHL & LISKOV 85] W.E. WEIHL et B. LISKOV
"Implementation of Resilient Atomic Data Types"
ACM Transactions on Programming Languages and Systems,
Vol.7,No.2 (avril-1985) pp.244-269

FORMULARIOS ELETRONICOS: DEFINIÇÃO E MANIPULAÇÃO AUTOMATICA DA INTERFACE DE USUARIO

José Palazzo Moreira de Oliveira - Duncan Dubugras Alcoba Ruiz
Universidade Federal do Rio Grande do Sul
Porto Alegre - *Brasil*

RESUMO

E' apresentado o conceito de Formulário Eletrônico bem como um modelo conceitual para a definição estrutural de formatação e de restituição de formulários. O mecanismo de geração da interface de usuário é descrito com sua possibilidade de apresentação automática de formulários eletrônicos. Este projeto está sendo desenvolvido , no Curso de Pós-graduação em Ciência da Computação da Universidade Federal do Rio Grande do Sul.

Palavras-chave: automação de escritórios, formulários eletrônicos, banco de dados generalizados, interface de usuário, programa especialista.

1. Introdução

Os Sistemas de Informação tem sido utilizados, na maior parte das vezes, em atividades ligadas à administração tradicional. Estes sistemas são apoiados por SGBDs convencionais baseados em modelos que descrevem e manipulam dados simples e pouco estruturados tais como inteiros, reais e cadeias de caracteres. Recentemente tem sido desenvolvidos esforços para integrar à estes sistemas outros grupos de atividades com características particulares. Estas novas áreas de aplicação apresentam necessidades diferenciadas de manipulação de dados.

Como consequencia, o desenvolvimento de novas pesquisas no campo de Banco de Dados tem-se direcionado, principalmente, para a inclusão de uma maior parcela da semântica da realidade no modelo de dados. Surge, então, a necessidade de dispor de novos tipos de, dados associados à esta extensão da semântica. Estes estudos levaram ao desenvolvimento de Sistemas de Gerência de Bancos de Dados Generalizados (SGBDG). A **Automação de Escritórios** é uma das áreas que mais tem-se destacado nas aplicações de SGBDG. A AE trabalha com os tipos de dados documentos, formulários e tempo.

Dentro da automação de escritórios um dos aspectos mais importantes é o uso dos formulários como elementos de transferência de informação e de controle. O formulário é o elemento privilegiado para a transferência de dados dentro de uma organização. Os formulários tem, também, um papel preponderante na estrutura de controle das operações. Este controle é processado pelo desencadeamento de atividades pela chegada de um formulário a uma estação de trabalho satisfazendo certas

condições pré-estabelecidas baseadas em seu conteudo.

Dentro de uma visão conceitual, um formulário é um tipo particular de documento generalizado [BRV 83]. No presente trabalho, entretanto, devido a sua importância particular o tipo formulário será considerado como um tipo independente, isto é, tendo definição própria. Os conceitos desenvolvidos para os documentos foram adaptados ao caso particular dos formulários.

2. Os Formulários Eletrônicos

O objetivo de um sistema integrado de formulários é a eliminação completa, dentro do ambiente de uma organização, do formulário impresso sobre papel.

Os formulários são um elemento comum nas atividades cotidianas. Eles permitem o intercâmbio ordenado e não ambiguo de informações. Estas características são devidas, principalmente, a sua estrutura ao mesmo tempo estruturada e de facil interpretação. Além disto as convenções para o uso de formulários são bem conhecidas e de uso universal. Por estes motivos os formulários são largamente utilizados nos sistemas de aplicações de AE.

Em um sistema informatizado, os formulários existem e são armazenados de forma digital. Estes formulários são denominados **Formulários Eletrônicos**.

O presente projeto integra os trabalhos sobre Modelos de Dados com aqueles ligados a AE e visa o estudo dos diferentes aspectos envolvidos na utilização de formulários em um ambiente automatizado. Um objetivo importante a ser atingido é a integração dos conceitos desenvolvidos em modelos de dados generalizados ([Cod 79], [SNF 80], [HM 81], [KL 82], [ACO 85]) nas aplicações de escritório.

O primeiro objetivo consiste em garantir a integração dos conceitos desenvolvidos em SGBDG com as atividades de escritório de forma a manter os procedimentos naturais de manipulação de formulários. Este é um fator importante pois da transição suave, sem rupturas dos procedimentos atualmente adotados, depende a aceitação de um sistema orientado ao usuário final.

O segundo consiste em desenvolver uma estrutura conceitual consistente para a definição dos diferentes níveis de especificação de formulários. Uma definição clara dos níveis de representação de formulários é essencial para assegurar a independência de dados e de formas de apresentação.

3. A arquitetura em níveis

O conceito básico utilizado para a definição de um modelo conceitual para a definição e manipulação de formulários é a decomposição desta atividade em um conjunto de níveis. Cada nível é caracterizado por suas funções e permite a decomposição do problema de modelagem em diversos sub-conjuntos isolados.

Este modelo em níveis foi proposto, inicialmente, para a representação de documentos em [BRV 83]. O mesmo permite uma grande independência de dados e de dispositivos físicos na modelagem e desenvolvimento de sistemas de aplicação. Uma das suas principais vantagens é a independência entre a definição das operações sobre o esquema (operadores de tipo) e sobre as ocorrências de formulários.

A figura 1 apresenta a arquitetura em níveis e as etapas do processo de criação de um formulário. A geração de um formulário é composta pela descrição do tipo de formulário, de suas características de formatação, pela criação de um formulário eletrônico e, finalmente, pela reprodução de um formulário em um meio físico.

Um grupo de formulários que possuam a mesma estrutura é identificado pelo correspondente **tipo de formulário**. O conceito de tipo de formulário é análogo ao de tipo de dados tal como empregado em linguagens de programação. A descrição do tipo de formulário é realizada atravez da linguagem de definição de dados associada ao **Modelo Sintático**.

A geração de uma ocorrência de um formulário corresponde à associação de um um conjunto de dados (**Informações Textuais**) ao modelo sintático . Isto é representado, no banco de dados, pela inicialização de uma área e pelo armazenamento dos dados correspondentes à estrutura definida no modelo sintático. Uma ocorrência de um conjunto de dados associada a um modelo sintático é denominada **Formulário Eletrônico**.

Um formulário possue características que não são associadas ao seu conteudo mas ligadas à forma de apresentação. A associação dos formulários eletrônicos as regras de apresentação lógica denomina-se **formatação**. Estas regras de apresentação são por exemplo: realce de um campo ou representação de um grupo de campos associados.

As regras de apresentação podem ser divididas em dois grupos: as independentes das restrições impostas pelas dimensões lógicas das páginas e aquelas vinculadas ao formato da página ou janela de exibição. Por exemplo: um texto poder ser formatado em duas colunas (parte puramente lógica) e estas colunas devem ser representadas em uma página com 96 caracteres de largura.

A regras de formatação são lógicas e, portanto, independentes do sistema físico de restituição. Sua modificação não altera o conteudo do formulário e, dependendo da aplicação,

um mesmo formulário eletrônico pode ser associado a diferentes conjuntos de regras de formatação.

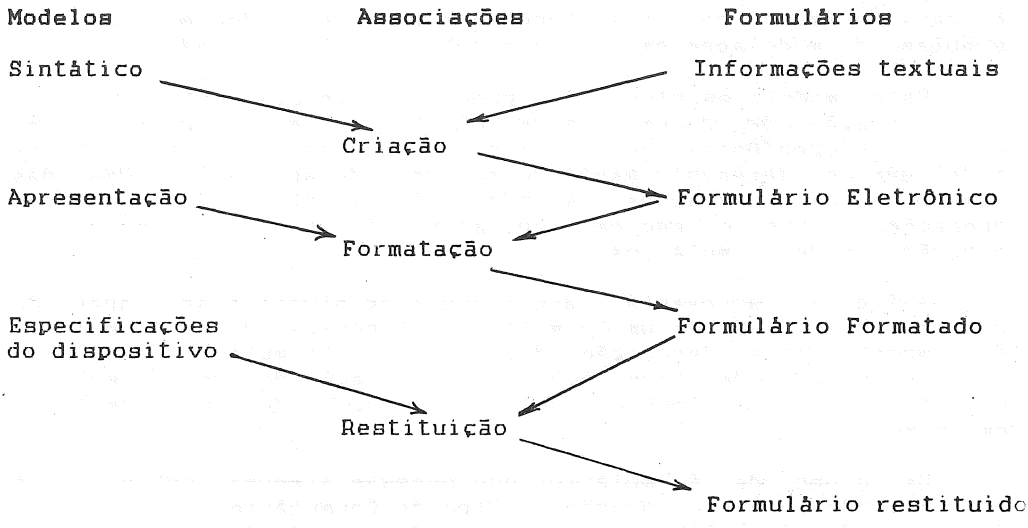


Fig 1 O modelo hierárquico para a representação de formulários.

Para ser percebido por um usuário o formulário formatado deve ser materializado em um dispositivo de saida. Esta atividade é denomimada **restituição**. Um formulário é **restituido** de acordo com as especificações do equipamento (tipo de caracteres, video inverso ou negrito etc). A restituição consiste na interpretação das regras de apresentação em função das caracteristicas fisicas do equipamento utilizado.

4. O modelo sintático de formulário

A descrição do tipo de formulário é feita pelo **Modelo Sintático**. Um modelo sintático constitui-se na descrição de um tipo de formulário. Um formulário é composto por um **cabeçalho** que é utilizado para a descrição externa e pelo **corpo**.

O corpo de um formulário é definido pela enumeração de suas propriedades. Cada propriedade é representada por meio de um campo definido sobre um tipo de dado semelhante aos da linguagem Pascal. Estes tipos correspondem ao conceito de **tipos simples**, **tipo renomeado** ou **tipo estruturado** [Pal 84].

Um tipo de formulário contem campos e grupos de campos, definidos sobre tipos de dados. Os grupos campos podem ser considerados entidades caracteristicas tais como descritas em

[Cod-79] e utilizadas para a representação de Formulários Abstratos em [Col 85].

As diversas operações disponíveis para a definição dos tipos de formulários são constituidas por operadores atuando sobre um ou mais tipos. Os operadores de tipo permitem a definição de formulários, a partir dos tipos de base ou de tipos anteriormente definidos.

A definição sintática de um formulário compreende ainda os **textos de fundo** (cabeçalhos, titulos de grupos de atributos, instruções e valores possiveis de atributos) e **parâmetros**.

Os identificadores de tipos são incluidos em um dicionário de dados onde, além da definição correspondente, são armazenadas informações sobre o uso, origem e outros dados administrativos. Desta forma e' possivel manter a uniformidade de descrição dos campos dos diversos tipos de formulário.

O processo de definição dos formulários é facilitado pela existência de uma função de apoio (help) que permite, a qualquer momento, a obtenção de informações sobre os campos dos formulários.

4.1 Declaração de um Formulário:

A declaração de um formulário é a associação de um identificador com uma definição de dados. Esta estrutura define todas as características do formulário e é suficiente para permitir sua restituição e manipulação.

Exemplo:

```
TYPE
     MOD70 [ATUALIZACAO DO CADASTRO DE FITAS]:
                                        <tipo-formulario>;
```

Sintaxe associada:

```
<declaracao-tipo-formulario> ::= <identificador>
                '['<titulo>']:' <tipo-formulario>
```

O <identificador> obedece as mesmas regras da linguagem PASCAL. O <titulo> corresponde ao nome externo do formulário.

4.2 Definição de um Formulário:

O elemento sintático <tipo-formulario> engloba toda a especificação dos elementos constantes de um formulário que constituem o "corpo do formulário". Estes elementos pertencem a três categorias: estrutura, apresentação e operações.

Exemplo:

```
...:Formulario
        estrutura

           <conjunto-elementos-estrutura>

        fim;

        apresentacao

           <conjunto-apresentacoes>

        fim;

           <def-operações>

     fim;
```

Sintaxe associada:

```
<tipo-formulario> ::= 'Formulario' <def-formulario>
                                                    'fim'

<def-formulario> ::= <def-estrutura>
                     <def-operacoes> <def-apresentacao>

<def-estrutura> ::= 'estrutura'
                      <conjunto-elementos-estrutura> 'fim;'

<def-apresentacao> ::= apresentacao'
                           <conjunto-apresentacoes> 'fim;'
```

Na <def-estrutura> é feita a definição do tipo formulário, ou seja, os campos constantes do formulário sendo definido.

Na <def-apresentacao> é feita a especificação de como os mecanismos de formatação e restituição devem agir a nível de interface de usuário.

Na <def-operacoes> é realizada a definição das operações que podem ser executadas sobre o formulário.

A definição do tipo em <def-estrutura> obedece aos mesmos mecanismos disponíveis na linguagem PASCAL existindo também operadores de tipo definidos em <tipo-derivado>. Estes modelos refletem as necessidades de manipulação de dados do usuário, ou seja, é definido um modelo que represente com fidelidade a realidade modelada.

Exemplo:

```
...: Formulario
        estrutura
           nro-serie [NRO. SERIE]:String(6);
           tamanho[TAMANHO]:('P','M','G');
           nome-resp[NOME DO RESPONSAVEL]:String(20);
              ...
        fim;
         ...
     fim;
```

Sintaxe associada:

```
<conjunto-elementos-estrutura> ::= <elemento-estrutura> |
          <elemento-estrutura> ';' <conjunto-elementos-estrutura>

<elemento-estrutura> ::= 'lista (' <min> ',' <max> ') de '
                         <declaração-elemento> |
                        'caso' <conjunto-casos> 'fim' |
                         <identificador-tipo-formulario> '.estrutura' |
                         <declaracao-elemento>
```

Através de <elemento-estrutura>, é possível criar novos formulários utilizando definições já existentes, estipular um intervalo do número de ocorrências possíveis e, também possibilidades sintáticas alternativas para um mesmo formulário (caso).

```
<declaracao-elemento> ::= <identificador '['<titulo']:'
                                                <elemento>

<elemento> ::= <tipo-simples> | <tipo-restrito> |
                  <tipo-composto> | <tipo-derivado>

<tipo-simples> ::= 'Inteiro' | 'Real' | 'Booleano' |
                   'String (' <inteiro-sem-sinal> ')'
```

A <declaracao-elemento> é semelhante à declaracao de variáveis na linguagem PASCAL. A diferença existente é a associação de um <titulo> ao identificador do campo para servir de identificação do campo ao usuário.

Os <tipo-simples> possuem características de armazenamento, manipulação e formatação padrões e, por conseguinte, não possuem especificação de número de posições (exceto os do tipo String) e restrições de integridade: são assumidas as características do equipamento no qual os formulários serão manipulados.

A definição de <tipo-restrito> engloba dois tipos de dados existentes na PASCAL:

```
<tipo-restrito> ::= <tipo-escalar> | <tipo-intervalo>

<tipo-escalar> ::= '(' <conjunto-elementos-escalares> ')'

<conjunto-elementos-escalares> ::= <elemento-escalar> |
        <elemento-escalar> ',' <conjunto-elementos-escalares>

<elemento-escalar> ::= <string> | <constante-numerica>

<tipo-intervalo> ::= '(' <constante-numerica> '..'
                                        <constante-numerica> ')'
```

A utilização desses tipos obedece os mesmos padrões da linguagem PASCAL.

```
<tipo-composto> ::= <tipo-arranjo> | <tipo-registro>

<tipo-arranjo> ::= 'arranjo (' <inteiro-sem-sinal>
                                    ') de ' <tipo-basico>

<tipo-registro> ::= 'registro' <conjunto-tipos-basicos>
                                                    'fim'

<conjunto-tipos-basicos> ::= <declaracao-tipo-basico> |
          <declaracao-tipo-basico> ';' <conjunto-tipos-basicos>
```

O uso de <tipo-composto> torna extremamente flexível a definição de formulários incluindo tabelas de campos e hierarquias.

```
TYPE
  MOD30[ATUALIZACAO DE HORARIOS]:
    Formulario
      estrutura
        cod-func[CODIGO FUNCIONARIO]:num-dc-func;
        transacao[TRANSACAO]: (1..3);
        mes-ref[HORARIO DO MES]:('JAN','FEV','MAR','ABR',
          'MAI','JUN','JUL','AGO','SET','OUT','NOV','DEZ');
        data-preench[DATA]:data6;
        regime[REGIME]:(1,2);
        benef-sem[BENEFICIO SEMANAL]:hora4;
        carga-hor[CARGA HORARIA CONTRATUAL]:hora4;
        horario-intencao[HORARIO DE INTENCAO]:
          arranjo (14) de horario;
        horario-obrigatorio[HORARIO OBRIGATORIO]:
          arranjo (14) de horario;
      fim;
```

onde:

```
type
  horario:
    registro
      inicio[INICIO]:hora4;
      duracao[DURACAO]:hora3-duracao;
      dia[DIA]:(2..7);
    fim;
```

Neste exemplo, ficam claras as vantagens da utilização de tipos como os <tipo-registro>s e <tipo-composto>s.

5. O modelo de apresentação

Nesta estrutura sintática é feita a especificação de parâmetros de formatação e textos para orientação do usuário a nível de manipulação via interface de usuário.

Exemplo:

```
apresentacao
   DOCUMENTAMOD70:
      globais
         cabecalho moldura enfatizado [UFRGS CPD];
      fim;
      nodo nro-serie: esquerda moldura;
      nodo tamanho: centrado moldura
            #P - PEQUENO, M - MEDIO, G - GRANDE#;
      nodo nome-resp: moldura;
         ...
   fim;
fim;
```

Sintaxe associada:

Para:

```
apresentacao
   DOCUMENTAMOD70:
      <construtor-apresentacao>
   fim;
fim;
```

```
<def-apresentacao> ::= 'apresentacao'
     <conjunto-apresentacoes> 'fim;'

<conjunto-apresentacoes> ::= <apresentacao> |
     <apresentacao> ';' <conjunto-apresentacoes>

<apresentacao> ::= <identificador> ':'
     <construtor-apresentacao>
```

Em cada <instancia-apresentacao>, é especificada uma forma de localizar e representar os campos. Um mesmo tipo formulário pode ter uma a "n" apresentações diferentes. Cada representação estará associada a, pelo menos, uma operação sobre o formulário. A vinculação das apresentações com as operações é feita em <lista-operacoes>.

Exemplo:

```
DOCUMENTAMOD70:
   globais
      cabecalho <forma-apresentacao> [UFRGS CPD];
   fim;
    ...
fim;
```

Sintaxe associada:

```
<construtor-apresentacao> ::= 'globais'
                  <conjunto-apresentacoes-globais> 'fim;'
                  <conjunto-apresentacoes-nodos>

<conjunto-apresentacoes-globais>::=<apresentacao-global> |
  <apresentacao-global>';'<conjunto-apresentacoes-globais>

<apresentacao-global> ::= <apresentacao-cabecalhos>
                                    <apresentacao-textos>

<apresentacao-cabecalhos> ::= <empty> | 'cabecalho'
                    <forma-apresentacao> '[' <texto> '];'

<apresentacao-textos> ::= <empty> | <conteudo-texto>
```

Em <apresentacao-global>, são definidas sentenças que titulam o formulário (<apresentacao-cabecalhos>) e textos orientadores ao usuário (<apresentacao-textos>). Os textos orientadores são visíveis ao usuário via os mecanismos de "help".

Exemplo:

```
DOCUMENTAMOD70:
    ...
    nodo nro-serie: <def-posicao> <forma-apresentacao>;
    nodo tamanho: <def-posicao> <forma-apresentacao>
                  #P - PEQUENO, M - MEDIO, G - GRANDE#;
    nodo nome-resp: <def-posicao> <forma-apresentacao>;
```

Sintaxe associada:

```
<conjunto-apresentacoes-nodos> ::= <apresentacao-nodo> |
    <apresentacao-nodo> ';' <conjunto-apresentacoes-nodos>

<apresentacao-nodo> ::= 'nodo' <nome-nodo> ':'
                        <def-posicao> <forma-apresentacao>
                                      <apresentacao-textos>
```

Na <apresentacao-nodo> são especificadas orientações à interface de usuário a respeito de posicionamento e apresentação. Em <conteudo-texto> são definidas orientações ao usuário sobre o campo, acessíveis através dos mecanismos de "help".

Sendo que:

```
<def-posicao> ::= <empty> | 'centrado' | 'direita' |
                                         'esquerda'

<forma-apresentacao> ::= <contorno> <apresentacao>

<contorno> ::=  <empty> | 'moldura'

<apresentacao> ::= <empty> | 'secreto' | 'enfatizado'

<conteudo-texto> ::= '#' <texto> '#'

<texto> ::= {sequencia de caracteres exceto ']' e '#'}
```

A <def-posicao> orienta a interface de usuário quanto à disposição do campo no plano de representação do formulário; a <forma-apresentacao>, quanto ao padrão de apresentação (formatação e restituição).

6. O processo de formatação e de restituição

A partir da definição sintática e das opções escolhidas na definição do modelo de apresentação é possível formatar os dados representados por um formulário eletrônico e posteriormente restituir o formulário formatado em um meio físico. O processo de restituição é vinculado ao dispositivo físico utilizado e

consiste no mapeamento dos dados para uma estrutura física bidimensional equivalente a um formulário tradicional sobre uma folha de papel.

A atividade de formatação é equivalente à realizada, nos sistemas tradicionais, pelo analista de formulários. O "lay-out" de um formulário é definido a partir dos dados a serem manipulados e é orientado por uma série de regras adotadas por uma organização ou regras estéticas que levam a uma aparência agradável e que propiciam uma utilização facilitata.

A aptidão para desenvolver este trabalho é adquirida ao longo de um aprendizado lento e quase sempre não metódico. Dependendo muito de qualidades artesanais é um processo laborioso e de resultados não assegurados.

Pode ser traçado um paralelo entre a realização do "lay-out" de um formulário e a datilografia de um texto. A separação das palavras para permitir o ajustamento da margem direita é uma tarefa para datilógrafas bem treinadas. Os editores de texto simples resolvem este problema incuindo espaços entre as palavras e hifenizando alguma palavra mais longa. Desta forma a tarefa de justifição é automatizada liberando o digitador das decisões ligadas a apresentação do texto.

Entretanto as decisões tomadas por uma boa datilógrafa não se limitam ao conteudo de uma linha mas levam em conta toda a aparência de uma página. Neste caso as diferentes decisões elementares são dependentes das demais decisões já tomadas, isto é, as decisões são dependentes do contexto.

Em um sistema automatizado é possivel representar-se as diferentes regras de formatação e o processo decisório dependente do contexto de tal forma que cada página apresente a melhor configuração de acordo com certos critérios estéticos. Este método é utilizado pelo editor COBATEF [PJN-85]. O conjunto de regras e de procedimentos necessários para realizar esta tarefa formam um sistema especialista que realiza o processo de decisão que de outra forma seria desenvolvido pelo redator do documento.

No caso de formulários eletrônicos a tranferência destas atividades de geração do "lay-out" para um sistema especialista pode gerar uma melhoria considerável na qualidade do interface de usuário. A economia de tempo é também apreciavel pois esta constitui-se em uma fase longa na concepção de um formulário. Por outro lado a possibilidade de serem realizadas operações de consulta envolvendo dois ou mais formulários eletrônicos e gerando um novo tipo, não definido no esquema, apresenta uma necessidade de geração automática de "lay-outs".

Em um sistema que permite a manipulação de formulários de forma não limitada aos tipos definidos no esquema a possibilidade de formatação automática é essencial. Os tipos novos são gerados

por meio de operações sobre os tipos já existentes criando novos tipos para os quais não existem definições, em forma explícita, das regras de formatação.

6.1 Arquitetura do gerenciador do interface

A arquitetura do gerenciador de interface foi desenvolvida a partitir dos conceitos empregados para a construção de sistemas especialistas. O conhecimento necessário para a execusão da sua atividade é representada por um conjunto de regras que são interpretadas para produzir a configuração do formulário a ser restituido.

Cada possibilidade de formatação está sendo codificada em forma de regras que permitem selecionar uma ação a partir do estado parcial da restituição do formulário e das diferentes possibilidades de apresentação disponiveis para uma determinada estrutura de dados. As regras tem a seguinte estrutura:

<estado_form_X> & <estrt_dados_A> --> <rotina_Y>,<coef_mérito>

onde **<coef_mérito>** é um coeficiente indicando uma avaliação quantitativa sobre a qualidade da solução obtida pela execussão da **<rotina_Y>**. A arquitetura deste sistema é representada na figura 2.

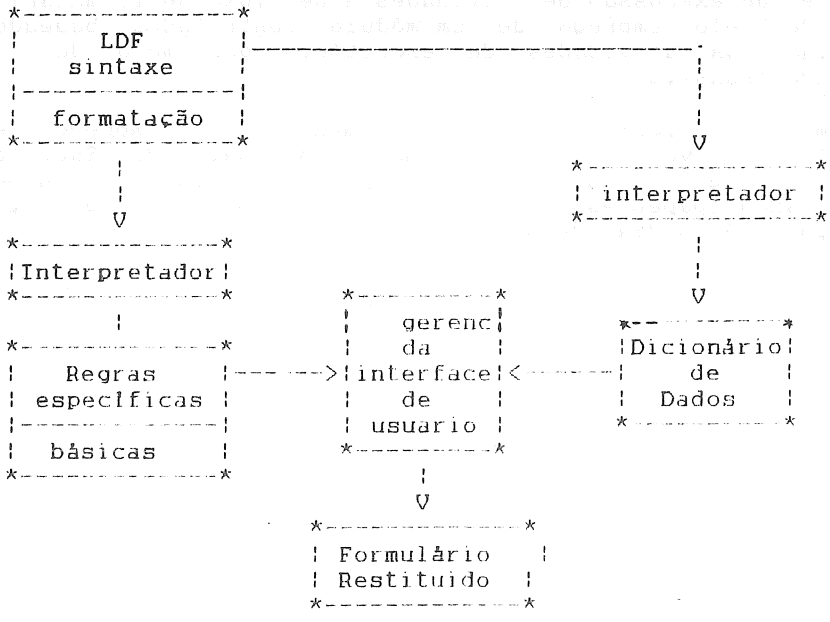


Fig 2: Arquitetura do sistema de restituição

Para um determinado formulário, a definição de formatação permite o interpretador da Linguagem de Definição incluir novas regras à base de conhecimentos. Estas regras estarão ligadas a exclusivamente este tipo de formulário. Quanto interpretadas, em conjunto com as regras básicas, permitem a formatação completa do formulário.

7. Conclusões

Em nossos estudos sobre Automação de Escritórios, tornou-se evidente que o sucesso de um sistema integrado de AE está intimamente ligado a possibilidade de automatizar as funções de apoio liberando as pessoas para as atividades criativas. A solução trivial de automatizar as tarefas de baixo nivel sem ser exploradas as possibilidades evoluidas da automação levam a sistemas pouco integrados que produzem resultados fracos para o aumento da eficácia do trabalho.

O projeto de Formulários Eletrônicos [Pal 86] destina-se ao desenvolvimento, tanto no plano conceitual quanto no prático, de soluções para atividades de AE. O acesso aos recursos distribuidos de computação é obtido por uma rede de microcomputadores e por um servidor de formulários. A interface de usuário é gerenciada por um sistema especialista. Finalmente o controle de execussão de atividades e de fluxo de formulários é garantido pelo emprego de um módulo controlador baseado na descrição das atividades de escritório por meio de redes predicado-transição.

Em nosso ponto de vista, somente pelo emprego destas metodologias evoluidas os sistemas de Automação de Escritórios poderão atingir o nivel de difusão e aceitação imaginados a alguns anos (paperless-office) e não confirmado pela evolução da automação até os dias de hoje.

BIBLIOGRAFIA

[BRV 83] G. Bogo, H. Richy, I. Vatton, "Proposition de Modèle pour la Normalisation des Documents", Relatório de pesquisa TIGRE n 3, março 83, Institut National Polytechnique de Grenoble, Franca.

[Cod 79] EF Codd, "Extending the Relational Database Model to Capture more Meaning", ACM Tranasctions on Database Systems, v 4, n 4, pag ???, dez 79.

[Col 85] C. Collet, "Les Formulaires Multi-Media", Séminaire INFORSID '85, Luchon, França, mai 1985.

[HM 81] M. Hammer & D McLeod, "Database Description with SMD: a Semantic Database Model", ACM Transactions on Dadtabase Systems, v 6, n 3, pag 351, set 81.

[KL 82] I. Kowalski & M. Lopez, "The Document Concept in a Database", Proceedings da ACM SIGMOD Conference, Orlando, Flórida, USA, 82.

[Pal 84] J. Palazzo Oliveira, "Le Modèle de Données et sa Représentation Relationnelle dans un Systeme de Gestion de Bases de Données Generalisées", Tese de Doutorado, Institut National Polytecnique de Grenoble, 1984, França.

[Pal 86] J. Palazzo Oliveira, "Electronic Forms, project description.", Relatório de pesquisa CPGCC/UFRGS n 47, jun 86.

[PJN-85] A. Peels, N. Jansen & W. Nawijn, "Document Architecture and Text Editing", ACM TOIS, V 3, N 4, OCT 85, pag 347.

[SNF 80] C.S. Santos, E.J. Neuhold & A.L. Furtado, "A Data-type Approach to the Entity-Relationship Model", em Entity Relationship Approach to System Analysis and Design, ed P.P. Chen, North Holland, 80.

[Tsi 82] D. Tsichritzis, "Form Management", Communications of ACM, v 25, n 7, jul 82.

[YHS 84] S.B. Yao, A.R. Hevner & Z. Shi, "Formanager: an Office Forms Management System", ACM Transactions on Office Information Systems, v 2, n 3, pag 453.

UM SISTEMA PARA CLASSIFICAÇÃO SUPERVISIONADA DE SILHUETAS EM IMAGENS BINARIAS

Jacob Scharcanski
Rômulo Silva de Oliveira
Universidade Federal do Rio Grande do Sul
Porto Alegre - *Brasil*

RESUMO:

O presente trabalho apresenta um sistema de classificação supervisionada de silhuetas em imagens binárias, desenvolvido na UFRGS no quadro de uma pesquisa em reconhecimento de padrões aplicado à controle de processos. São apresentadas as características relevantes da estrutura e funcionamento do sistema, e por fim, são propostas algumas aplicações.

1. INIRODUÇÃO

Atualmente existe um grande interesse sobre a área de Análise de Imagens para fins Industriais e Militares. Nas próximas décadas, espera-se que esta área adquira ainda maior importância, conforme indicam algumas aplicações listadas abaixo /BRAD 82/ :

- reconhecimento e aquisição de objetos automaticamente;
- direcionamento automático para ferramentas de corte e soldagem;
- processos relacionados com VLSI (Very Large Scale Integration), tais como, conectar pinos ou encapsular pastilhas;
- prover realimentação visual para a montagem e manutenção automática;
- inspeção de circuitos impressos quanto a curtos, erros ou más conecções;
- checagem dos resultados em processos de fundição, quanto a fraturas e impurezas.

O Sistema de Reconhecimento e Aprendizado de Silhuetas em Imagens Binárias proposto, enquadra-se na área de Processos Industriais, mais especificamente, no reconhecimento e aquisição automática de objetos diferenciáveis por suas silhuetas.

2. RECONHECIMENIO DE PADRÕES E ANÁLISE DE CENAS

Existe uma diversidade considerável de abordagens à análise de informações visuais por computador, portanto as fronteiras entre as diferentes visões tornam-se frequentemente vagas. Antes de detalhar o sistema proposto, vamos localizá-lo em um contexto, sem no entanto seguir um rigor formal, como não é nossa preocupação neste trabalho.

No âmbito da Análise de Cenas, a preocupação básica é a interpretação ou compreensão de imagens; existe uma intersecção de interesses com o Reconhecimento de Padrões, mas diferentes abrangências (ou abordagens). Exemplifiquemos:

- Sistemas de Reconhecimento de Padrões tipicamente reconhecem uma entrada dentro de um conjunto limitado (geralmente pequeno) de possibilidades. Na Análise de Cenas tenta-se construir descrições ricas para cada imagem individualmente, é o caso quando deseja-se computar objetos tridimensionais, e não reconhecê-los como instâncias de um certo número de protótipos armazenados;
- Sistemas de Reconhecimento de Padrões estão mais relacionados à imagens bidimensionais, tais como símbolos gráficos. A abordagem a objetos tridimensionais, como é o caso de peças mecânicas, é feita efetivamente tratando-os como bidimensionais, considerando cada posição estável do mesmo como um objeto separado. Já a Análise de Cenas trata extensivamente de objetos tridimensionais (stereo, representação);
- de uma forma geral, Sistemas de Reconhecimento de Padões tipicamente operam diretamente sobre a imagem. Os processos visuais da análise de cenas operam, não sobre a imagem, mas sobre representações simbólicas computadas anteriormente a partir da imagem, como por exemplo visão stereo, análise de textura ou inferência de forma a partir do sombreamento.

O presente Sistema de Reconhecimento e Aprendizado de Silhuetas em Imagens Binárias enquadra-se na área de Reconhecimento de Padrões.

3. O SISIEMA PARA CLASSIFICAÇÃO SUPERVISIONADA DE SILHUEIAS EM IMAGENS BINÁRIAS

3.1 O OBJEIIVO DO SISIEMA

O presente sistema tem por objetivo dar uma solução ao problema de reconhecer vários objetos através de suas silhuetas, utilizando a configuração mostrada na figura 3.1.

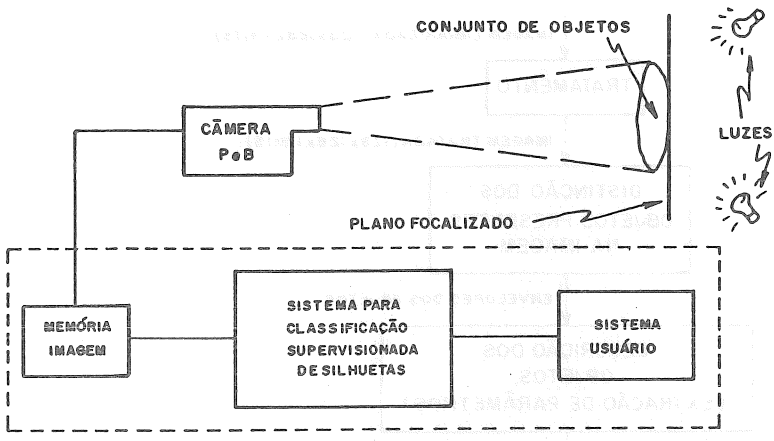


Figura 3.1

O sistema proposto interage com um "sistema usuário", que precisa da informação tipo do objeto submetido para tomar decisões.

As imagens dos objetos a reconhecer são adquiridas por uma câmera de tv, em preto e branco. Os objetos desfilam sobre um plano iluminado conforme mostra a figura 3.1 , e a distância focal da câmera está ajustada para este plano. Quando o conjunto de objetos atinge o centro do plano focalizado, é obtida uma "fotografia" (imagem estática) do mesmo.

A imagem assim obtida é então submetida ao sistema de classificação supervisionada para que classifique-a e entregue a informação TIPO DO OBJETO ao "sistema usuário".

3.2 A DESCRIÇÃO DO SISIEMA

Para melhor descrevermos o sistema e seu funcionamento, vamos separar seu aspecto "organização" da "forma de operação".

3.2.1 ORGANIZAÇÃO

O sistema é composto por módulos que operam sobre a imagem, visando reduzir a quantidade de informação a processar, ou sobre a representação da imagem, para identificar, segundo critérios estabelecidos, a silhueta de um objeto, ou a classe a que pertence. A figura 3.2 apresenta os módulos e seu relacionamento, definindo a organização do sistema.

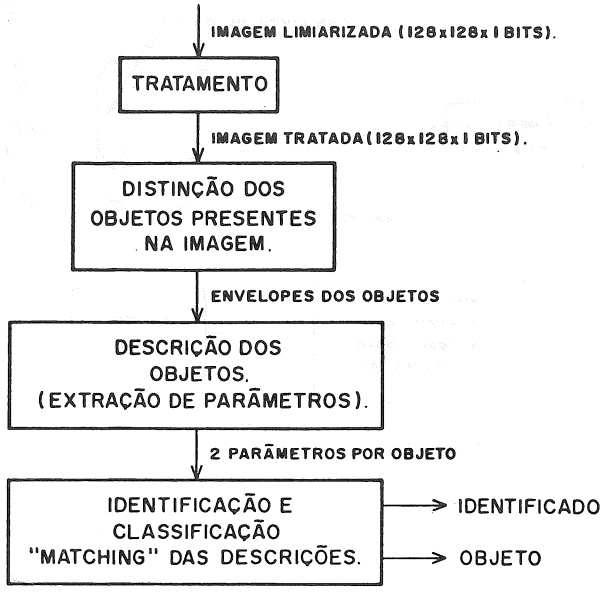


Figura 3.2

3.2.2 FORMA DE OPERAÇÃO

A imagem obtida pela câmera de tv, digitalizada no formato 128 x 128 elementos de imagem, com 1 bit cada, deve ser submetida a um tratamento para que a informação apresentada seja consistente, ou seja, procura-se eliminar da imagem possíveis ruídos (informações não significativas), restando apenas regiões homogêneas em um dos dois níveis: regiões escuras-0, ou regiões claras-1. Para a redução do ruído presente na imagem, existem vários métodos e algorítmos propostos, como por exemplo, os métodos apresentados em /HALL 79/.

Uma vez obtidas as regiões homogêneas que compõe a imagem, é realizada a distinção das silhuetas dos objetos através do módulo de segmentação, permitindo assim classificar as silhuetas individualmente "a posteriori". O método utilizado na segmentação é concorrente e será apresentado na seção 3.3. O produto final da segmentação é uma lista de envelopes que determinam a localização dos objetos na imagem. A figura 3.3 apresenta o módulo de segmentação e seu relacionamento com o módulo de descrição.

Nesta etapa do processamento, a silhueta a identificar está bem definida, vamos reduzir a quantidade de informação

presente, descrevendo-a por dois parâmetros, que consideramos suficientes para diferenciar imagens em um conjunto limitado de formas. Estes parâmetros são:
a) a relação envelope/área da silhueta;
b) o desvio padrão em relação ao ponto médio da silhueta.

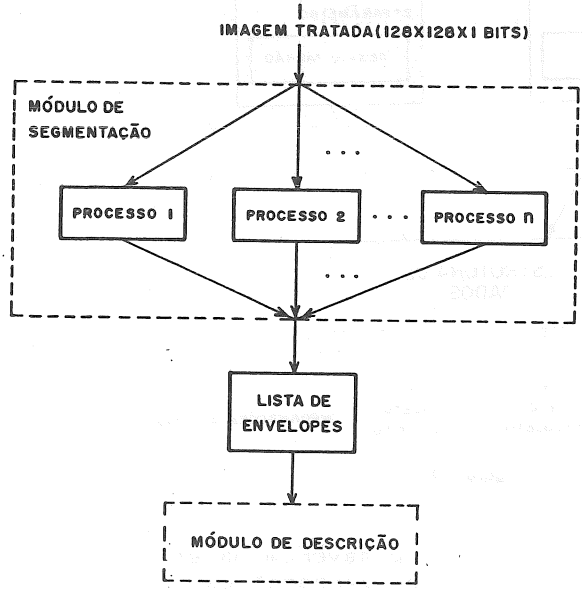


Figura 3.3

Observa-se que os critérios escolhidos são independentes da posição, orientação e escala dos objetos. Nesta etapa, houve uma "tradução" da imagem em dois números, resultantes da avaliação dos parâmetros acima definidos sobre a imagem. A figura 3.4 apresenta o módulo de descrição e seu relacionamento com o módulo de identificação e classificação.

A identificação do TIPO DO OBJETO é feita no MÓDULO DE IDENTIFICAÇÃO E CLASSIFICAÇÃO, como mostra a figura 3.4, por algoritmos que utilizam os dois parâmetros citados para tentar encontrar na estrutura de dados algum objeto que se enquadre nesta descrição. Se o objeto "fotografado" for NÃO IDENTIFICADO, não foi encontrada na estrutura de dados nenhuma descrição que se adapte a este objeto, o "sistema usuário" pode solicitar que o NOVO OBJETO seja classificado e sua descrição armazenada na estrutura de dados, para ser reconhecido posteriormente.

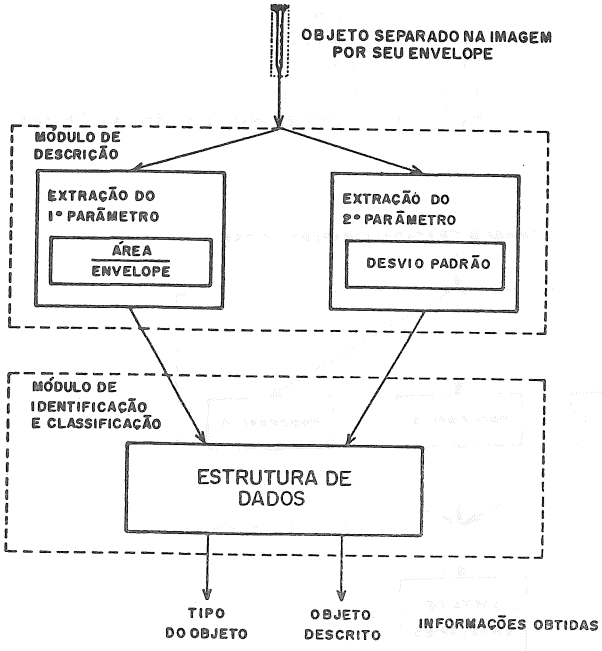


Figura 3.4

Pode ocorrer o caso de haverem objetos "semelhantes", como será mostrado quando for apresentada a estrutura de dados, no momento basta dizer que dois objetos serão "semelhantes" quando o primeiro parâmetro de ambos for igual (dentro de uma certa tolerância), e houver apenas um objeto, cujo primeiro parâmetro tenha este valor. Outro objeto submetido ao sistema durante a fase de "aprendizado" ou treinamento, que tenha a mesma relação área/envelope, será identificado inicialmente por semelhança, podendo o usuário recusar esta identificação e optar por classificá-lo e armazená-lo como um NOVO OBJETO na estrutura de dados; então será usado como critério de diferenciação entre ambos o segundo parâmetro. Se este objeto for novamente apresentado ao sistema, este "lembrará" do objeto e saberá diferenciá-lo do seu semelhante (pelo segundo parâmetro).

Tendo em vista o que foi apresentado, o sistema deve operar em dois modos distintos:

- treinamento: etapa em que são apresentados os objetos a serem reconhecidos "a posteriori". O sistema

os classificará e armazenará na estrutura de dados se isto lhe for solicitado;

- utilização: etapa em que o sistema está sendo solicitado pelo "sistema usuário" a fornecer a informação TIPOS DOS OBJETOS que lhe são submetidos. Os objetos foram previamente classificados e armazenados na estrutura de dados (na etapa de aprendizado). Neste modo, o sistema, ou reconhecerá o objeto e entregará a informação TIPO DO OBJETO, ou não o reconhecerá e o considerará NÃO IDENTIFICADO.

3.3 O MÉTODO CONCORRENTE DE SEGMENTAÇÃO

Entendemos por segmentação, a extração ou identificação dos objetos contidos em uma imagem, onde objeto é toda característica com conteúdo semântico relevante para a aplicação desejada. Em nosso caso, os objetos são regiões homogêneas (silhuetas), e consideramos segmentado o objeto que tem seu envelope (menor retângulo que o contém) determinado na imagem binária. Assumimos que os objetos tem envelopes não sobrepostos. Vamos apresentar o método concorrente de segmentação de uma maneira introdutória, maiores detalhes podem ser encontrados em /OLIV 86/.

Inicialmente, a matriz binária que representa a imagem, é percorrida segundo a varredura até que seja encontrado um primeiro elemento de imagem pertencente a um objeto, o contorno de sua silhueta é determinado, assim como o seu envelope.

Uma vez conhecido o envelope deste primeiro objeto, a matriz é subdividida logicamente em cinco áreas, como mostra a figura 3.5. A área 1 é aquela que foi percorrida inicialmente e sabe-se que nela não existe objeto algum, pode ser ignorada. A area 2 compreende o envelope do objeto determinado, como os envelopes são disjuntos, ou seja, não há sobreposição, esta área também pode ser ignorada. Portanto, a existência de objetos ainda não determinados é possível apenas nas áreas 3,4 e 5, onde irá prosseguir a pesquisa. A cada uma das três áreas por pesquisar, agora menores que a área inicial, os procedimentos acima descritos são reaplicados de forma independente entre si. É da reaplicação simultânea do método que provêm a natureza concorrente do mesmo. Existem situações específicas em que algumas áreas, ou mesmo todas, podem não existir, que não serão tratadas nesta apresentação introdutória. No caso de existirem outros objetos, outras áreas serão delimitadas, ocorre então áreas divididas em áreas menores , e estas por sua vez também divididas, até que toda a matriz seja pesquisada.

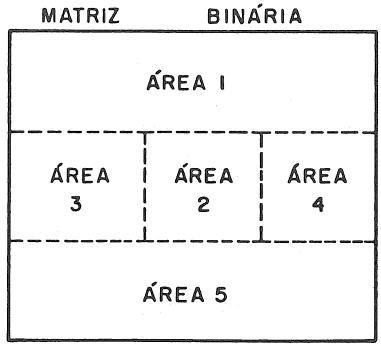


Figura 3.5

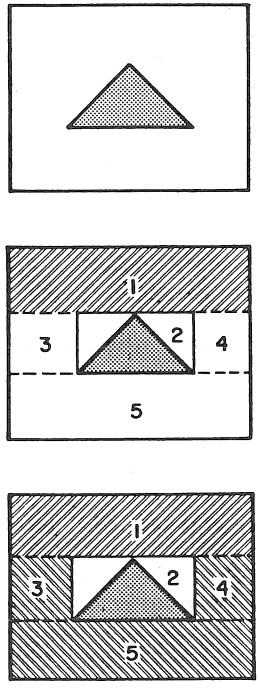


Figura 3.6

O processo da divisão da imagem em áreas, e em áreas

ainda menores, não considera a existência de outros objetos além daquele que causa a divisão. Podem ocorrer situações em que objetos são particionados por divisões feitas na imagem, fazendo parte de duas ou mais áreas distintas; estas e outras situações são tratadas com detalhe em /OLIV 86/.

Ao término da pesquisa completa da matriz obtém-se uma lista contendo os envelopes de todos os objetos encontrados, para serem identificados e/ou classificados "a posteriori".

Tipicamente, as funções realizadas pelo método descrito encontrarão aplicação em sistemas de tempo real. Isto significa que o método terá valor efetivamente se for possível implementá-lo de forma eficiente em termos de tempo de execução. A figura 3.6 apresenta a aplicação do método apresentado a uma imagem binária composta de um único objeto, por simplicidade.

3.4 A ESTRUTURA DE DADOS

Neste sistema a estrutura de dados é de importância fundamental, por isto dedicaremos atenção a ela. Para suportar as especificações do sistema quanto a diferenciação dos objetos, organizamos a estrutura de dados de forma hierárquica, permitindo assim o armazenamento das descrições dos objetos e posterior identificação utilizando um número mínimo de parâmetros, reduzindo, por consequência, o "overhead" de pesquisa. Para isto foram utilizadas como chaves de acesso os dois parâmetros obtidos na descrição da imagem, conforme ilustra a figura 3.7.

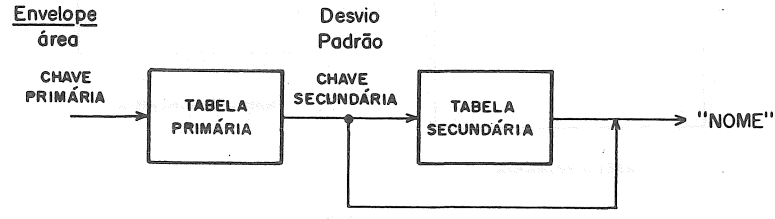


Figura 3.7

O conjunto de informações armazenado nas tabelas é o necessário para relacionar a descrição de um objeto com o seu "nome" (a sua identificação). Pode-se ter descrições mais "ricas" do objeto utilizando mais parâmetros, mas neste sistema, como já foi salientado, utilizamos dois parâmetros para descrevê-lo, por este motivo temos duas tabelas compondo a estrutura de dados, para N parâmetros teríamos N tabelas. O

conteúdo destas tabelas está representado na figura 3.8.

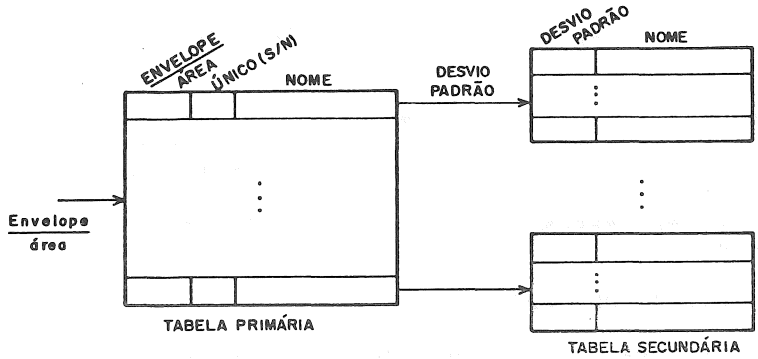


Figura 3.8

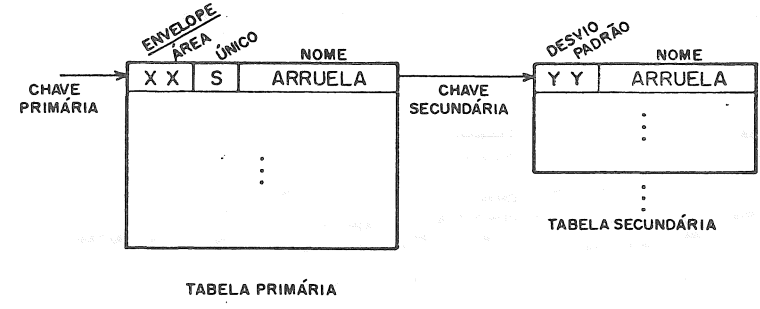


Figura 3.9

O ideal seria organizar a estrutura de dados para uma busca otimizada, mas por simplicidade preferiu-se organizá-la colocando as chaves por ordem de inserção e fazer busca sequencial.

3.4.1 ACESSO NO MODO IREINAMENIO

Para cada objeto submetido ao sistema é verificado se a sua descrição está presente na estrutura de dados, então existem duas hipóteses:

- objeto não presente:

O NOVO OBJETO é armazenado na estrutura de dados como mostra a figura 3.9 .

- objeto presente:

Neste caso, pode ocorrer de haver objetos "semelhantes", ou seja, objetos cujo primeiro parâmetro está dentro de uma faixa de tolerância , especificada pela descrição do objeto (presente na estrutura de dados). Esta faixa de tolerância é determinada empiricamente, como função da aplicação que se tem em vista para o sistema, e do conjunto de formas que compreende cada classe de objetos. A "semelhança" de objetos refere-se à semelhança que há entre o primeiro parâmetro de uma descrição, presente na estrutura de dados e que é única, e o primeiro parâmetro de um objeto submetido ao sistema. O sistema apresenta o "nome" do objeto presente na estrutura de dados como identificação, caso o usuário não aceite esta resposta, o "nome" do NOVO OBJETO será armazenado na TABELA SECUNDÁRIA juntamente com a sua CHAVE SECUNDÁRIA de acesso, diferenciando os dois objetos, já que a CHAVE PRIMÁRIA permanece a mesma para ambos e o primeiro objeto já não é único. A figura 3.10 ilustra o estado final da transação.

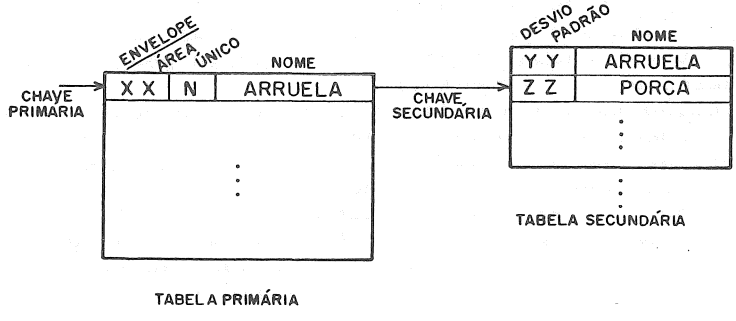


Figura 3.10

Para alterar o nome de um objeto já submetido e clas-

sificado, então basta submetê-lo novamente ao sistema e alterar o seu nome.

3.4.2 ACESSO NO MODO UTILIZAÇÃO

Neste modo, cada objeto submetido ao sistema será considerado:

- "IDENTIFICADO": Se a sua descrição "fecha" com a de algum objeto classificado presente na estrutura de dados. Neste caso, retorna o "nome" do objeto encontrado como a identificação para o objeto submetido;
- "NÃO IDENTIFICADO": Se a sua descrição não "fecha" com a de nenhum objeto classificado presente na estrutura de dados. Esta situação é notificada ao "sistema usuário", constituindo uma exceção ou falha do sistema, pois todos os objetos submetidos a esta altura deveriam ter sido classificados no modo "aprendizagem".

4. APLICAÇÕES

Como já foi salientado, o presente sistema aplica-se à classificação das formas de silhuetas em um contexto definido, ou seja, é capaz de identificar um objeto que lhe seja submetido utilizando um conhecimento, um conjunto de formas pré-classificadas; dentro deste âmbito existem varias aplicações. Para ilustrar podemos apresentar duas:

- pode-se utilizá-lo para reconhecer o código associado a um dado objeto, e informar o sistema de controle da identidade deste objeto e outras informações relevantes relacionadas a éste código, como é o caso no controle automático dos ítens produzidos ou estocados. Neste caso o código pode ser complexo (alfanumérico, por exemplo);
- uma aplicação mais avançada seria o emprego deste sistema, com um refinamento na aquisição de imagens, para orientar o deslocamento de robôs no ambiente industrial através de "imagens-código" reconhecidas visualmente, dispostas ao longo de um caminho a ser seguido /SCHA 85a/. Esta técnica proposta para alterar a posição de robôs em uma área de trabalho, permite reconfigurar o ambiente de uma indústria dinamicamente. Basta alterar os códigos para que a indústria passe a operar de maneira diferente.

Sem dúvida que a implementação do sistema em uma máquina sequencial é limitada em velocidade para o reconhecimento em tempo real; uma solução seria a utilização de uma configuração composta de diversos pontos de aquisição de imagens e uma máquina central especializada em processamento de

imagens para processá-las /SCHA 85a/; ou ainda, emular os algoritmos em "hardware".

5. CONCLUSÕES

O Sistema de Classificação Supervisionada de Silhuetas em Imagens Binárias apresentado oferece uma solução ao problema proposto, dentro de um contexto definido, como é característico dos sistemas de reconhecimento de padões.

Se por um lado a dependência de contexto é um fator limitante, por outro simplifica a construção de um sistema especializado eficiente, facilitando a emulação dos algoritmos em hardware, o que viabiliza a operação em tempo real.

Devido a modularidade do sistema, é possível a sua expansão, aumentando a estrutura de dados e o vetor de parâmetros que descrevem uma imagem.

A experiência adquirida no projeto deste sistema de classificação permite fazer certas recomendações, que serão de utilidade em seu aperfeiçoamento posterior, tais como:

a) O conjunto de N atributos de forma determina um espaço N dimensional, e as classes que compreendem os objetos são regiões deste espaço. Para descrever efetivamente objetos de forma mais complexa, é necessário um conjunto maior de atributos de forma, ou seja, descrever classes ou regiões num espaço com mais dimensões;
b) Para que a classificação seja eficiente, os atributos de forma devem ser escolhidos especificamente em função do conjunto de objetos que serão submetidos ao sistema;
c) Os atributos devem ser independentes entre si, para que não seja reduzida a separação entre as classes, inclusive, deve-se escolhê-los de maneira a maximizar a separação entre classes;
d) Ajustar a faixa de tolerância para cada atributo de uma classe por treinamento, utilizando os objetos pertencentes a esta classe;
e) Pode-se reduzir o tamanho da estrutura de dados descrevendo cada classe por um vetor de atributos, onde cada atributo tem sua faixa de tolerância especificada assim como o valor médio, obtidos na fase de treinamento; este conjunto de informações é suficiente para a classificação na fase de operação.

6. BIBLIOGRAFIA

/BRAD 82/ - "Computational Approaches to Image Understanding"
Michael Brady
Computer Surveys, vol.14, nº1, march, 1982.

/CAST 80/ - "Digital Image Processing"
Kenneth R. Castlemman
Prentice-Hall,Inc., 1980.

/FU 77/ - "Data Structures, Computer Graphics and Pattern Recognition"
K.S.Fu, A.Klinger , T.L.Kunii
Academic Press,Inc., 1977.

/HALL 79/ - "Computer Image Processing and Recognition"
Ernest L. Hall
Academic Press,Inc., 1979.

/MASC 84/ - "Processamento Digital de Imagens"
Nelson D.A. Mascarenhas, Flavio R.D. Velasco
IV Escola de Computação, São Paulo, 1984.

/OLIV 86/ - "Um Sistema Concorrente Para Distinguir Objetos em Imagens Binárias"
Rômulo S. de Oliveira, Jacob Scharcanski
Iº Semana de Informática da Universidade Federal da Bahia, 1986.

/SCHA 85a/- "SMP - Sistema Maciçamente Paralelo, Definição e Aplicações"
Jacob Scharcanski, Philippe Navaux
XVIII Congresso Nacional de Informática, 1985.

/SCHA 85b/- "Um Sistema de Reconhecimento e Aprendizado de Silhuetas em Imagens Binárias"
Jacob Scharcanski, Rômulo S. de Oliveira
2º Congresso Nacional de Automação Industrial, 1985.

GRUPO IV

PROLOG: UNA HERRAMIENTA IDEAL PARA EL DESARROLLO DE PROTOTIPOS?

Gustavo Arango Gaviria
Universidad de los Andes
Bogotá - *Colombia*

PROLOG : Una herramienta ideal para la investigación en la universidad (léase desarrollo de prototipos)
Su ilustración con una aplicación en bases de datos

0. GUIAS DE LECTURA.

El artículo se escribió pretendiendo tener varios tipos de lectores:

- quienes no conocen Prolog pero desean conocer algo de sus posibilidades (deben leer la descripción del sistema sin tratar ahondar en las explicaciones sobre las gramática ni en las cláusulas PROLOG que se presentan para implementar algunas de las características del sistema).
- quienes no conocen de teorías sobre analizadores que deben omitir todas las explicaciones técnicas al respecto.
- las personas familiarizadas con PROLOG, bases de datos y gramáticas a quienes todo debe interesarles.

I. Introducción, supuestos y motivaciones.

Se pretende ilustrar la tesis enunciada en el título a través de la descripción de un sistema de consulta en lenguaje natural para bases de datos definidas según el modelo entidad-relación.

Antes que nada es indispensable definir lo que es la investigación a nivel de informática en un departamento de sistemas donde ofrece - por el momento - exclusivamente un pro\ ma de pregrado.

Los profesores pueden solo dedicar una cuarta parte de su tiempo a proyectos de investigación puesto que gran parte de su trabajo está consumido por la docencia.
En la mayoría de los casos la generación de resultados proviene de los trabajos en tesis de grado que son componentes de proyectos dirigidos por profesores que trabajan en áreas de interés común definidas según sus especialidades.

Supuestos básicos:

a. Por la estructura misma de la universidad como generadora de conocimientos tenemos que la manera más eficaz de concretar los resultados de investigación es por medio de la generación de maquetas o prototipos.
Con esta forma de funcionar se logra :
- mostrar y difundir entre profesionales e industriales el alcance real de los proyectos de investigación que se llevan a cabo.
- no exigir un gasto exagerado de recursos, ni un compromiso largo de la investigación en problemas donde el interés primordial es mostrar la factibilidad de buenas soluciones y no la obtención de productos finales.

Aceptemos entonces que el desarrollo de prototipos es el principal producto de la investigación.

b. Aun si admitimos la posibilidad de generación por la universidad de productos de software; una de las etapas a lograr lo más pronto posible en el ciclo de vida de un programa verdaderamente operacional(ver :comercializable) es justamente un prototipo.

La obtención de un prototipo permite la interacción con el usuario cuyo aporte es fundamental para identificar:

- las fallas y omisiones en la comunicación hombre-máquina
- las indefiniciones en el problema,
- las fallas en el diseño.

c. Dada la gran difusión de micro-computadores, cada vez es mayor el número de sus utilizadores.
El diferente nivel de capacitación de estos frente a las máquinas exige gran flexibilidad de las aplicaciones .

El objetivo que se busca es llegar a exigir lo mínimo del usuario por parte de la aplicación.
Lo ideal es que el lenguage de la aplicación se acerque al máximo al lenguaje del usuario y no lo contrario.
Una de las posibilidades es usar el lenguaje natural o en su defecto manejar la ilusión de comunicar en lenguaje natural utilizando un subconjunto de éste.

El propósito de la aplicación que se describe es el de ilustrar las posibilidades de PROLOG para la generación de prototipos de aplicaciones donde la comunicación con el usuario se hace buscando acercarse al máximo a su lenguaje

II. Descripción del sistema

El sistema presentado es el resultado de dos meses de investigación en el período de descanso en la docencia, su meta era explorar hasta donde se podía llegar en la investigación en

Prolog, creando un sistema de multiples aplicaciones con algunas características (flexibilidad, facilidad de comunicación, verificación de coherencia de datos) que lo hicieran llamativo a usuarios finales (aquellos que no requieren de grandes conocimientos en informática para usar una aplicación)

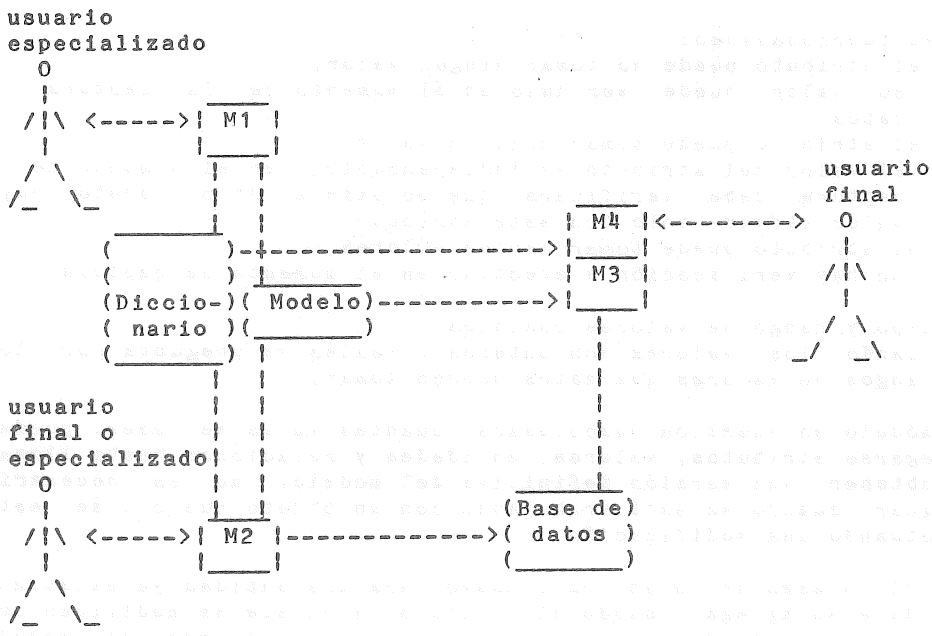
Se trata de un prototipo de un manejador de bases de datos relacionales que puede consultar el usuario final en un subconjunto del lenguaje natural.

Es importante hacer notar que el sistema que se describe necesita de un tiempo de implementación mucho mayor cuando se usa una herramienta clásica (un lenguaje de programación de alto nivel como Pascal o C).

El sistema comprende cuatro módulos:

- Definición del modelo entidad-relación (M1)
- Captura de los datos (M2)
- Sistema de consulta con base en predicados de Prolog (M3)
- Interfaz en lenguaje natural (M4)

DESCRIPCION GRAFICA DEL SISTEMA



LOS MODULOS:

- M1. Modelo entidad-relación
- M2. Captura de los datos
- M3. Sistema de consulta en Prolog
- M4. Interfaz en lenguaje natural

CONVENCIONES:

(ARCHIVO) |PROGRAMA|

A.Definición del modelo entidad-relación.(M1)

El sistema pide al usuario la descripción de cada uno de los objetos de trabajo : entidades, atributos o relaciones.
A medida que se obtienen nuevos nombres de objetos se actualiza un diccionario que permite la posibilidad de uso de sinóminos y pregunta por las formas en plural de cada item del diccionario

En forma interactiva el programa permite obtener:
- las entidades y sus atributos,
- las relaciones, relaciones que intervienen y sus atributos.

Para todo atributo se determina:

- si se trata de la clave o parte de ésta,

- su funcionalidad:
  el atributo puede no tomar ningun valor,
  (su valor puede ser nulo en el momento de la captura de datos)
  el atributo puede tomar solo un valor
  (el valor del atributo es indispensable y en el momento de la captura debe verificarse que no existe otro objeto cuyo valor difiere solo con este atributo)
  el atributo puede tomar varios valores
  (no hay verificación a efectuar en el momento de captura)

- tipo y rango de valores posibles:
  cuando los valores son enteros o reales se pregunta por los rangos de valores que estos pueden tomar.

El módulo en cuestión puede usarse cuantas veces se desee: pueden agregarse atributos, valores, entidades y relaciones hasta llegar a obtener una versión definitiva del modelo, no es necesario indicar cuándo se está trabajando con un objeto nuevo o se está efectuando una modificación.

vg: Al presentar un atributo nuevo para una entidad ya existente éste se agrega; cuando el atributo ya existe se modifican sus características sin que esto cause traumatismos al modelo existente.

La implementación del modelo entidad-relación es inmediata, cada objeto del modelo se representa como un hecho PROLOG.

Por ejemplo la información sobre una entidad se hace:

```
entidad(Código de su nombre, Número de atributos,
        Lista de los códigos de sus atributos llaves)
```

haciendo abstracción de los códigos tendríamos:

```
entidad(estudiante,4,[carnet])
```

B.Captura de datos.(M2)

Se pregunta al usuario cual es el nombre del objeto sobre el que quiere dar información.

Mediante menús se obtiene:

- para entidades:
  los valores debidamente validados (según se explica en el modulo anterior) de sus atributos.

- para relaciones:
  los valores validados de sus atributos,
  los valores validados de los atributos necesarios para caracterizar las entidades que intervienen.

(Se considera que los atributos que conforman la llave de una entidad identifican claramente; cuando no hay llaves es necesaria la información sobre los valores de todos los atributos)

Ejemplo:

si se definieron las entidades:

- estudiante
  .atributos :
    nombre, carnet, teléfono y programa
  .llave:
   carnet

- materia
  .atributos :
   nombre, código, departamento
  .llave:
   código

y la relación:

- vio
  entidades:
   estudiante, materia
  atributos:
   nota, semestre

entonces al hacerse la captura de datos sobre la relación se pregunta por las informaciones correspondientes a:
- carnet [llave de estudiante]
- código [llave de materia]
- nota y semestre (atributos de la relación)

Cuando un valor no es numérico, se codifica y se agrega a una tabla de los valores posibles de un atributo dado.
Así se permite cuando el usuario lo desee investigar sobre los valores posibles que puede tomar un atributo.
Esta opción no fue incorporada en el sistema pero su programación es inmediata usando el predicado setof (descrito en [CLO 81])

Ejemplo:

en la base de datos:
- fruta(limon)
- fruta(pera),
- fruta(uva),

pregunta:
- setof(X,fruta(X),L),

respuesta:
- L = [limon, pera, uva]

C. La consulta en PROLOG.(M3)

Se han considerado por el momento las preguntas de información directa sobre un objeto :

- para entidades:
  - preguntar por algunos o todos sus atributos,
  - con o sin restricciones sobre los valores de otros de sus atributos

ie: estudiantes de nombre Pedro Pérez
(respuesta: todos los atributos de estudiantes de nombre Pedro Pérez)

carnet de los estudiantes de nombre Pedro Pérez
(respuesta: solamente el valor del carnet)

teléfono y carnet de los estudiantes de ingenieria
(respuesta:carnet y teléfono de los matriculados en ingeniería)

- para relaciones:
  - preguntar por algunas de sus entidades o de sus atributos,
  - con o sin restricciones sobre los valores de algunas de sus entidades o de sus atributos

ie: estudiantes que vieron la materia 21117
(respuesta :carnet de los estudiantes que vieron la materia de código 21117 y todos los valores de los atributos de la relacion vio)

nota de las materias que vió el estudiante 8080800
(repuesta : código y nota de las materias que tienen como identificación para la entidad estudiante el carnet 8080800)

D. la interfaz en lenguaje natural.(M4)

Es quizá en este módulo donde se aplican las técnicas más avanzadas y donde se explotan mejor las características más potentes de PROLOG

La parte fundamental es un generador de analizadores sintáctico-sémantico que recibe la grámatica y produce un programa Prolog que será el que haga efectivo el análisis de una serie de tokens (formas normalizadas de las declinaciones-conjugaciones de las palabras en español) y produce el predicado PROLOG que corresponde a la pregunta del usuario.

El proceso de análisis tiene dos partes :

- Una general independiente de las consultas que requiere de un usuario especializado para que defina la grámatica en su lenguaje
(ésto es prácticamente lo único que debería modificarse para responder consultas en otros lenguajes)
Una vez definida la gramática debe utilizarse el módulo que la lee y genera el programa PROLOG que hace el análisis.
Esto debe repetirse cada vez que se cambie la gramática, pero la labor del usuario especializado se limita a escribir adicciones o modificaciones de las reglas de producción que la componen.

- El análisis de una consulta específica consta de :

  - análisis léxico que recibe la frase del usuario y produce la lista de símbolos (tokens) que sirve de entrada al módulo siguiente,

  - análisis sintáctico-semántico que genera el predicado PROLOG que sirve de consulta.

A continuación se describen en detalle las componentes de proceso de análisis; primero la gramática y sus particularidades, posteriormente un ejemplo de lo escrito para la aplicación y finalmente el proceso general de consulta.

1.La gramática.

Está compuesta por reglas de producción cuya forma general es:

no_terminal ---> lista de terminales, de no terminales y/o de cláusulas semánticas

a.No terminales.

Un no_terminal es un término Prolog con variables y átomos y se puede visualizar en el momento del análisis como un predicado PROLOG que se encargará de hacer un análisis parcial de la cadena de entrada.

b.Terminales.

Un terminal puede tomar varias formas:

i- [atomo]

el análisis tiene éxito si se identifica el símbolo analizado en la cadena de entrada con atomo y se pasa al siguiente símbolo

ie: [ser]

ii- [{atomo}]

es opcional que figure atomo como símbolo de entrada en el análisis en cuyo caso se avanza al símbolo siguiente

ie: [{sobre}]

iii- [objeto^^Variable]

objeto : corresponde a uno de los elementos del lenguaje en el modelo entidad-relación (entidad, atributo, relación, ...)

el análisis tiene éxito cuando el símbolo de entrada pertenece a la categoría especificada por medio de objeto, se pasa al token siguiente y se instancia Variable con el valor con que se codificó en el diccionario.

ie: [entidad^^Var]
tiene éxito si el símbolo de entrada es materia,

[atributo^^Var]
tiene éxito si el símbolo de entrada es nota

(el valor de Var corresponde a los códigos asignados respectivamente por el sistema a materia y nota)

iv- [[Pos,predicado]]

- predicado corresponde a un 'functor' PROLOG y a un predicado

- Pos permite ligar la cadena de entrada con el argumento del 'functor' predicado con cuya variable se instanciará, Pos es la posición de la variable dentro del predicado;

el análisis tiene éxito cuando el símbolo analizado o una sublista de la cadena de entrada satisface predicado.

Con esta forma de terminal se logra permitir un look-ahead pues se puede examinar un numero variable de símbolos en la cadena de entrada.

ie:[[5,atributo(estudiante,nombre,_,_,Val)]]
funciona si:
- atributo(estudiante,nombre,x,y,'pedro perez') está en la base de datos
- pedro, perez son los siguientes símbolos a analizar en la cadena de entrada

c.Acciones semánticas

Se trata simplemente de predicados PROLOG definidos por el usuario para obtener resultados adicionales al análisis.
Como estos predicados deben diferenciarse de los no_terminales de la gramática se introduce un nuevo operador para indicar el cambio de interpretación.
Se utiliza el operador @ para introducir los predicados de PROLOG.

ejemplo:

las siguientes reglas permiten analizar y contestar preguntas sobre la hora actual:

```
responda_hora ---> tiempo(Hora,Minuto),
                   @ write('hora:'),write(Hora),
                     write('minuto:'),write(Minuto).

tiempo(Hora,Minuto) ---> [que], [hora] ,[[es]], [[?]]
                         @ time(time(Hora,Minuto,_,_)).
```

el predicado time(X) se responde consultando el reloj del sistema y tiene éxito instanciando X con el 'functor':
time(Valor_hora,Valor_minutos,
Valor_segundos,Valor_centesimas_de_segundo)

2.Ejemplo de parte de la gramática utilizada en la aplicación.

```
/* identificacion del símbolo distinguido */
    disting(sdist,4).

    sdist(Cad,Obj,Lval,Lres) --->
                    [[ sobre ]],
                    det_cad(Cad,Obj,Lval,Lres).

/* producción que identifica una pregunta sobre una entidad */
    det_cad(entidad,Enti,[Latval],[Latr]) --->
            atri(Latr),  /*  no terminal que identifica
                             restricciones sobre valores de atributos */
            [entidad^^Enti],   /* identificación de una entidad */
            busq_atr_val(Enti,Latval).  /*  no terminal que identifica
                                            los atributos sobre los que
                                             se desea informacion */

/* producciones que identifican el valor de un atributo */
    val_clav(Ent,Atri,Val0) --->
            [de], [atributo^^Atri],      /* identificación de un atributo */
            mire_val(Ent,Atri,Val,Val0).
    val_clav(Ent,Atri,Val) ---> mire_val(Ent,Atri,Val).
             /*  identificacion  de un  valor  de  atributo
                 cuando se omite su nombre */

    mire_val(Entidad,Atri,Val0) --->
       [[4, verifique_val_pos(atributo,Entidad,Atri,Val,Val0)]].
     /* identificación y validación del valor del atributo */

/* produciones que identifican una lista de valores*/
    lista_val_clav(Entres,[[Atr|Claval]|Lres]) --->
        [','],
        val_clav(Entres,Atr,Claval),/* identificación  de un valor
                                       de un atributo */
        lista_val_clav(Entres,Lres).

    lista_val_clav(Entres,[[Atr|Claval]]) --->
        [y], val_clav(Entres,Atr,Claval).

    lista_val_clav(Entres,[]) ---> lambda.
```

la cláusula:

```
disting(sdist,4)
```

permite identificar el símbolo distinguido de la gramática y su número de argumentos como 'functor' PROLOG,
esta información será aprovechada por el generador de analizadores para ceder el control al predicado indicado.

3.El análisis de una consulta.

a. El analizador léxico.

Se recibe una frase que se procesa convirtiendola en una lista de tokens. Todo blanco en la frase se considera un separador de símbolos.
Las palabras originales de la frase sufren transformaciones en la lista de tokens:

- hay una lista de palabras cuyo aporte semántico se considera irrelevante; toda palabra que pertenezca a esta lista se elimina de la lista de símbolos.

- se aprovechan las indicaciones dadas por el usuario al definir su modelo para transformar el plural o las formas conjugadas de los verbos y obtener una forma normal

  ie: estudiantes pasa a ser estudiante
  vieron se transforma en vio

b. El análisis semántico.

La lista de símbolos producida por el analizador léxico se usa como entrada al programa PROLOG generado a partir de la gramática por el generador de analizadores.

III. Ventajas de PROLOG en la aplicación.

A. Facilidades en el análisis.

1. No es necesario escribir un programa especial para leer la grámatica.

Las reglas de producción pueden leerse como términos PROLOG.
Es suficiente usar la facilidad de definición de operadores que tiene el lenguaje.
Usando el predicado op (ver [ARI 84], [CLO 81])

Es necesario definir :
- ^^ y ---> como operadores infijos,
- @ y { como operadores prefijos,
- } como operador sufijo.

2. La gramática es muy fácilmente modificable

Cualquier opción nueva necesita solamente de la adición, y modificación de las reglas de producción de la gramática.
El generador de analizadores se encarga del trabajo y por su potencia al incluir la posibilidad de look-ahead y de utilización de predicados PROLOG es una herramienta que no debe ser modificada.

Es necesario anotar aquí una de las características de PROLOG que puede pasar desapercibida:

la posibilidad de generar por medio de un programa PROLOG otro al cúal pueda cedersele el control

esto se logra mediante el uso de los predicados call, functor arg y =.. que no son necesariamente parte de todas las implementaciones existentes del lenguaje como la versión de TURBO-PROLOG (ver [TUR 86])

R.Facilidades de generalización.

La aplicación que trabaja sobre el modelo entidad-relacion es general en cuanto al tipo de objetos que el usuario puede definir en su base de datos.

Es en cuanto al idioma escogido para hacer las preguntas que hay que efectuar modificaciones sobre el programa.

Prolog permite manejar las palabras redundantes, las palabras del metalenguaje, y los sinónimos en forma de tablas lo que los hace fáciles de identificar y de modificar.

A continuación se presentan algunas de estas tablas en la aplicación:

```
/* tabla con algunas de las palabras redundantes al hacer
 preguntas en espanol */

 sino(informacion).          sino(el).
 sino(aparecer).             sino(dar).
 sino(lo).                   sino(haya).
 sino(decir).                sino(cuyo).
 sino(saber).                sino(conocer).
 sino(mostrar).              sino(indicar).

/* tabla de formas normales de palabras del lenguaje usado para
  hacer preguntas en espanol */

 sinon_l(saber,[sabe,saben,sabes,sepas,sepan,sepa]).
 sinon_l(conocer,[conoce,conoces,conocen,conozca]).
 sinon_l(listar,[liste,listeme,listenos]).
 sinon_l(lo,[aquello,aquella,aquellas,aquellos]).
 sinon_l(el,[los,la,las]).
 sinon_l(dar,[deme,denos]).
 sinon_l(decir,[diga,digame,diganos]).
 sinon_l(indicar,[indiqueme,indiquenos,indique]).
 sinon_l(informacion,[informaciones]).
 sinon_l(mostrar,[muestreme,muestrenos,muestre]).
 sinon_l(ser,[es,son]).
 sinon_l(cuyo,[cuya,cuyas,cuyos]).
```

No se pensó en lo mismo con los mensajes al usuario y es entonces necesario inspeccionar todo el código para cambiar de idioma en la comunicación que parte del sistema hacia el usuario.

Sinembargo la declarativiдad de prolog permite facilmente solucionar este problema para facilitar su modificacion:

ie: write('desea dar informacion sobre'), write(X)
donde X es un elemento del lenguaje del modelo entidad-relación

puede transformarse en:
write_meta(prompt_info),write_meta(X)

el predicado write_meta se puede definir:

```
write_meta(Cad):- tabla_prompt_(Cad,Cad1), !, write(Cad1).
write_meta(Cad):- write(Cad).
```

como parte de tabla_prompt, se tendría el hecho:

```
tabla_prompt(prompt_info,'desea dar informacion sobre').
```

La introducción de la tabla localiza en un solo sitio los cambios que habrían de hacerse, a nivel de preguntas del sistema, cuando se cambia de idioma.

C.Facilidad para encontrar la respuesta a una pregunta.

La idea del módulo de respuesta se resume en:

- identificar el objeto sobre el que se pregunta y por ende su representación PROLOG
- identificar las características que debe cumplir este objeto; los valores que deben tomar algunos de sus atributos. Esto se logra con una lista de posición y valor para cada uno de los atributos.
- construir un patrón de respuesta que se unificará - haciendo uso de la unificación de PROLOG - con los hechos correspondientes en la base de datos.

Los argumentos de entrada utilizados por este módulo son obtenidos por el programa de análisis. Como se verá a continuación la programación del patrón de respuesta es muy simple y su uso es general.

El predicado siguente permite construirlo :

```
/* patron_resp(Objeto,Argumentos,Lista_restricciones,Patrón)
    Objeto  :  nombre en la base de datos del objeto por el que se
               pregunta
    Argumentos : Número de elementos del objeto en cuestión

    Lista_de_restricciones : lista de parejas (posición, valor)
        posición: sitio donde se representa el atributo
                  restringido
        valor: el valor que el atributo debe tomar

    Patrón : Patrón de Respuesta a buscar en la base de datos  */

 patron_resp(Obj,Arg,L,P):-
             functor(P,Obj,Arg),
             llene_functor(L,P).

 llene_functor([],P):-!.
 llene_functor([[Pos|Valor]|Resto_L],P):-
                          arg(Pos,P,Valor),
                          llene_functor(Resto_L,P).
```

(los predicados arg y functor son los standard PROLOG)

D.Facilidades de consulta.

Cuando se hace un proceso repetitivo donde no es necesario guardar información sobre lo que pasa en cada una de las iteraciones, PROLOG permite un esquema de trabajo muy simple y efectivo.
Este esquema se usa para generar las repetidas preguntas del usuario.

Las cláusulas PROLOG que siguen ilustran la forma de realizar la consulta:

```
preguntas :- repeat,
       write('escriba su pregunta'),nl,
       analice,  /* predicado que se ocupa de todos
                    los pormenores de una consulta
                    solo se satisface una vez */
       mire_siga.

mire_siga:- write('hay mas preguntas ?'),
            escriba_s_n, !, fail.
mire_siga:- !.

escriba_s_n:- write('conteste:(s/n)'),
              get(Z),
              low_case(Z,Z1), /* transforma un caracter mayuscula
                                 en minuscula todo otro caracter
                                 lo deja igual */
              mire_resp(Z1),!.

/* clausulas para manejar la contestacion s/n */
 mire_resp(Z):- name(s,[Z]),!.
 mire_resp(Z):- name(n,[Z]),!, fail.
 mire_resp(_):- escriba_s_n. /* repite el proceso en caso de
                                respuesta no contemplada */
```

Como puede verse el predicado mire_resp:

- falla cuando el usuario responde con 'n'. En este caso mire_siga tendrá éxito y el predicado preguntas terminará correctamente.

- tiene éxito cuando el usuario responde con 's' lo que genera un nuevo intento de contestación; como repeat siempre puede recontestarse, se genera nuevamente el proceso.

La combinación repeat-fail es recomendada por muchos de los implementadores de PROLOG ([ART 84],[EXP 83], [TUR 86]) porqué permite un ahorro importante de espacio y es muy sencilla de utilizar para describir procesos iterativos cuyas iteraciones son independientes entre sí.

IV. MEJORAS Y LIMITACIONES.

Es claro que la aplicación descrita dista mucho de un producto terminado pero sirve para conocer pronto qué tan lejos se puede llegar, y para ilustrar algunas de las facilidades y técnicas para escribir en PROLOG procesos que no estamos acostumbrados a usar por la falta de un mecanismo de unificación o de re_ensayo (vg: la combinación repeat-fail)

Otros de las limitaciones del sistema presentado son :

- una gramática que solo contempla algunas pocas construcciones sintácticas del español
  (fácilmente corregible aumentandola)

- solo en algunos casos particulares se pueden usar correctamente las locuciones y grupos de palabras como unidades que constituyen un solo símbolo :

  - acerca de
    debe tomarse como 'acerca de' sinónimo de 'sobre'
    y no como los dos símbolos: acerca, de

  - atributo de relación
    no debe ser una serie de tres símbolos sino un objeto del lenguaje del modelo entidad-relación que es en este caso el atributo de una relación.

  (aquí la corrección se obtiene cambiando el módulo de análisis léxico, e incluyendo una tabla con las palabras del lenguaje usado para hacer preguntas y para describir los objetos que conforman el modelo del modelo entidad-relación)

El reto y la importancia de PROLOG se centran en que el paso del prototipo esbozado - aquel que contiene todas las mejoras presentadas a lo largo de este artículo - al prototipo implementado es rápido y sencillo.
La implementación efectiva del nuevo prototipo se hizo en una semana, la aplicación ganó en generalidad y en rapidez de análisis.
Es conveniente, a la luz de lo presentado, evaluar la tesis inicial. El prototipo de una aplicación no trivial se obtuvo con rapidez, las pruebas y modificaciones de éste permitieron efectuar la construcción de uno nuevo y en general el proceso de desarrollo se agilizó.

A pesar de los muy buenos resultados es importante hacer notar que no se trabajaron problemas fundamentales de paquetes de software como son el manejo de gran cantidad de información, y las facilidades en manejo de archivos.

Los ejemplos nos muestran la efectividad de lenguaje PROLOG en campos como el manejo de símbolos, la consulta de bases de datos y la generación de interfaces hombre-máquina.

BIBLIOGRAFIA

[ARI 84] Arity/Prolog interpreter Version 3.2
Arity Corporation.

[CLO 81] W. F. Clocksin y C. S. Mellish
Programming in Prolog,
Springer-Verlag, 1981

[EXP 83] MS-DOS Prolog Version 2.1
Expert Systems Ltd.
Oxford, U,K.

[TUR 86] Turbo-Prolog Owner's Handbook
Borland

CALCULO DE PROGRAMAS

Una metodología precisa para el diseño de programas de computador

Jaime A. Bohórquez V.
Universidad de Los Andes
Bogotá - *Colombia*

Resumen: Se presenta la metodología de diseño de programas de Gries y Dijkstra desde el punto de vista de estrategias de Solución de problemas. Se ilustra esta visión desarrollando completamente un ejemplo.

Palabras Claves: calculo de programas, corrección total, refinamiento a pasos, programación estructurada, aserciones.

0.Introducción

La programación de computadores debe considerarse como una forma de solución de problemas. Más precisamente, dado un problema bien especificado, se quiere construír un programa que lo solucione. Esta relación no se debe perder de vista al desarrollar (construír) cualquier programa, y debe siempre tenerse en cuenta que los métodos de programación están emparentados con los métodos (constructivos) de resolución de problemas.

Entre los pioneros en el desarrollo de una metodología de programación se encuentra Wirth [W71], quien es uno de los propulsores del método de refinamiento a pasos. Tales ideas son más ampliamente precisadas y fundamentadas por Hoare [H69], Dijkstra [Dj76] y Gries [G81] quienes proponen un cálculo de programas "guiado por objetivos" que consiste en una construcción simultánea del programa y la prueba de su corrección.

Desde otro punto de vista se encuentra a Jackson [J75], quien muestra las ventajas obtenidas al hacer corresponder la estructura de un programa con la de sus datos.

Aunque la presentación y estilo de estos autores es bien diferente, lo que en principio podría sugerir que se trata de enfoques independientes, en realidad existe una gran coherencia en sus trabajos. En la metodología de programación sugerida por Gries se reconoce una correspondencia entre las estructuras de control primitivas de un lenguaje algorítmico con las estrategias de solución de problemas utilizadas para diseñar programas; en el desarrollo de Jackson se encuentra una correspondencia análoga entre las mismas estructuras de control utilizadas por Gries y Dijkstra y las estructuras de datos que deben manejar los progra-

mas.

En este artículo se presenta el lenguaje algorítmico LCG (Lenguaje de Comandos Guardados) de Dijkstra como un instrumento para implementar una estrategia general de solución de problemas. La conexión del lenguaje LCG con estrategias de solución de problemas se entiende entre líneas en la literatura, pero el autor no conoce ninguna publicación en donde sea presentada explícitamente y con la importancia que se merece. (Algo en este sentido se encuentra en Hehner [Hh79], aunque allí la atención se centra en la proposición de un lenguaje algorítmico que excluye el comando iterativo, tal vez la pieza central del desarrollo de programas en LCG.)

Se hará énfasis en la aplicación práctica de una metodología de solución de problemas como herramienta de programación. La semántica de los comandos será introducida informalmente. Formulaciones precisas de los comandos LCG y su significado (semántica axiomática) se encuentran en [H69] y [Dj76].

1. El calculo de programas de Dijkstra y Gries.

La metodología de Dijkstra y Gries para el diseño de programas puede interpretarse en una estrategia de solución de problemas. Esta consiste inicialmente en la especificación del resultado deseado como una aserción formulada en principio en el cálculo de predicados (muchas veces es suficiente una enunciación precisa en lenguaje natural).

Obtenidas las pre- y post-condiciones que describen el problema, el lenguaje de programación se usa como un conjunto de comandos que efectuan transformaciones sobre predicados, que se denotan de la siguiente manera

$$\{Q\}\ S\ \{R\}$$

(léase el programa S es totalmente correcto con respecto a Q y R)

En esta expresión Q representa una precondición que es cierta antes de la ejecución del comando S, y R representa una post-condición que es verdad después de la ejecución del comando S. Es decir la formula significa que si la ejecución de S inicia en un estado que satisface Q, entonces terminará en un estado que satisface R.

La semántica de los comandos del lenguaje de programación concebido por Dijkstra está en correspondencia con tres métodos de solución de problemas:

a) Dividir y Conquistar (Comando de secuenciación)
   Dada la precondición Q y el objetivo R, establezca primero un subobjetivo R_1 mediante un comando (o programa) S_1 es decir

$$\{Q\}\ S_1\ \{R_1\}$$

luego, basandose en el logro de R_1, mediante un segundo comando S_2 establezca R, es decir S_2 es tal que

$$\{R_1\}\ S_2\ \{R\}$$

es cierto. Por tanto el programa que soluciona el problema original es la secuenciación de los comandos S_1 y S_2 o sea

$$S = S_1;S_2$$

b) Reducción a casos (comando de Selección)
Si se puede encontrar predicados B_1, B_2,..., B_n tales que Q implique alguno de los B_is, es decir $Q \Rightarrow B_1$ o B_2 o ... o B_n, entonces para establecer R dada la precondición Q basta usar el comando de selección

$$\begin{array}{rll} IF = & \underline{if} & B_1 \text{ ---> } S_1 \\ & [] & B_2 \text{ ---> } S_2 \\ & & \ldots \\ & [] & B_n \text{ ---> } S_n \\ & \underline{fi} & \end{array}$$

válido cuando los B_is son expresiones booleanas llamadas guardas, y los S_is son comandos que cumplen

$$\{Q \text{ y } B_i\}\ S_i\ \{R\}$$

(Observe que las precondiciones de estos subobjetivos son más fuertes que la precondición original del programa).

El rectángulo "[]" sirve para separar alternativas no ordenadas.
Al entrar a ejecutar a IF, se escoge nodeterministicamente una guarda verdadera y se ejecuta el comando correspondiente.
La expresión B_1 ---> S_1 se puede leer como "en caso de que la guarda B_1 sea verdad se debe ejecutar el comando S_1".
Si ninguna guarda fuera verdadera para el estado inicial de ejecución de IF, el comando aborta.

c) Iteración (solución por transformación gradual)
El comando iteración tiene la forma

$$\begin{array}{rll} DO = & \underline{do} & B_1 \text{ ---> } S_1 \\ & [] & B_2 \text{ ---> } S_2 \\ & & \ldots \\ & [] & B_n \text{ ---> } S_n \\ & \underline{od} & \end{array}$$

La iteración dura mientras por lo menos una de las guardas sea verdad. Del conjunto de comandos con guardas verdaderas se selecciona (no determinísticamente) una para ejecutarla.

Cuando se decide solucionar un problema con precondición Q y objetivo (o post-condición) R mediante una iteración, en general se tiene una idea o plan aunque sea vago para lograrlo. Estas ideas son plasmadas y concretadas en una aserción o predicado P que en general expresa un logro parcial del objetivo del problema.

El predicado P se llama un invariante del ciclo iterativo y los pasos para construir la iteración son los siguientes:

1) Partiendo de un estado cualquiera que cumpla Q (la precondición), obtener mediante un subprograma (en general trivial) un estado que satisfaga P es decir, establecer

{Q} inic {P}

donde inic es el subprograma mencionado.

2) Comparar el invariante y el objetivo para dilucidar una condición B que junto con el invariante implique la postcondición R es decir, encontrar B tal que

P y B => R

La negación de esta condición (¬B) constituirá en general, la única guarda del ciclo.

3) Escribir el cuerpo del ciclo de tal manera que su ejecución preserve la verdad del invariante y a la vez progrese hacia el logro de la guarda o condición B (y por consiguiente el logro de R).

El siguiente esquema resume los 3 pasos anteriores:

```
{Q}
inic
{P}
do ¬B ---> {¬B y P}
             S
           {P}
od
{R}
```

Aquí S es el cuerpo del ciclo.
Es posible que en la tarea de mantener a P invariante durante la ejecución de la iteración, sea necesario considerar varios casos; es decir, descomponer ¬B en subcondiciones B_1, B_2, ..., B_n dando lugar a la forma general del comando de iteración DO.

Para garantizar la terminación de la ejecución del ciclo iterativo, se usa una función cota t (entera) que depende del estado de ejecución, y estima el número de iteraciones que aún deben ejecutarse para terminar el comando.

La función cota t debe cumplir las siguientes propiedades:

(i) P y B => t>0
es decir mientras dure el ciclo t debe ser positivo.

(ii) {P y B y t≤ß} S {t<ß}
es decir t debe decrecer con cada iteración.

Por lo anterior, t decrece y se mantiene positiva mientras dura el ciclo iterativo, debido a ésto el ciclo no puede prolongarse indefinidamente (el rango de t es bien fundado); ß es un valor constante y entero que no aparece en ninguna otra parte del programa.

Esta última condición (ii) constituye una guía para el diseño del cuerpo del ciclo en el paso (3): Mantener el invariante progresando hacia terminación.

Se tiene entonces, un lenguaje de programación que responde a una metodología de solución de problemas con tres estrategias básicas Dividir y Conquistar, Reducción a casos y Transfomación gradual apoyada en una propiedad invariante; las cuales corresponden a las operaciones básicas de secuenciación, selección e iteración que son aplicadas recurrentemente a los subproblemas que van surgiendo con su aplicación, hasta llegar a subproblemas de solución sencilla (†).

Se puede también añadir a las tres estrategias mencionadas arriba, la que corresponde a la acción de abstraer la solución de un problema como un subprograma o procedimiento, posponiendo su solución a una etapa posterior del diseño. De hecho, Hehner [Hh79] propone la abstracción de subproblemas como herramienta principal del desarollo de programas, como al alternativa a la estrategia de transformación gradual mencionada en el párrafo anterior.

Es interesante el hecho de que Jackson [J75] proponga esta misma trilogía para el tratamiento de estructuras de datos secuenciales logrando de esta manera programas que corresponden estructuralmente a la jerarquía semántica de sus datos. Mediante este mecanismo se obtienen programas muy facilmente modificables para mantenerse al día con la evolución y cambios requeridos por el usuario del programa.

Queda ilustrado de esta manera el "isomorfismo" que existe entre el lenguaje de comandos guardados y la estrategia de solución de problemas descrita anteriormente. Cabe agregar que los comandos de dicho lenguaje son versiones generalizadas de los usados por casi todos los lenguajes imperativos de programación y coinciden con los recomendados por las disciplina informal comunmente denominada programación estructurada.

(†) Como comando primitivo de transformación de estados se tiene por supuesto a la asignación; cuyos efectos se pueden describir en el calculo que se está presentando como {Q} x:=e {R} si y solamente si $Q \Rightarrow R^x_e$. Donde R^x_e denota la sustitución textual de cada instancia libre de x por la expresión e.

Estos hechos contrastan con los métodos tradicionales de la enseñanza de la programación que reducen la labor de programar al diestro manejo y comprensión de los diferentes comandos de un lenguaje de programación, en total desconección con una disciplina de solución de problemas; juzgando además, la bondad de un lenguaje de programación por la variedad y sofisticación de sus comandos básicos como factor preponderante.

2. El cálculo en la práctica.

A continuación se mostrará el uso de la metodología previamente descrita construyendo un programa para solucionar un problema de dificultad moderada que permitirá sin embargo, ilustrar cada uno de los puntos discutidos anteriormente.

El siguiente problema fué tomado de [W182].

Problema: Dados n enteros mayores que 1, (n entero positivo) calcular para cada uno de los datos sus primos más cercanos (Si dos primos son equidistantes a alguno de los datos ambos deben ser calculados.)

Especificación mediante aserciones

Precondición: Q: d(1),...,d(n) son enteros mayores que 1.
Postcondición: R: C(1),...,C(n) son conjuntos de números tales que C(i)={p: p es primo más cercano a d(i)} para i=1,...,n.

Construcción del programa

La solución (global) evidente para este problema es un ciclo iterativo cuyo invariante en la etapa i reza

P: $1 \leq i \leq n$ y C(j)={p: p es primo más cercano a d(j)} para j=1,..,i-1

Para no aburrir al lector con detalles de la aplicación de la metodología en esta primera fase, que es bastante evidente, se presenta sin más explicaciones la primera aproximación a la solución del problema.

```
i:=1;
do i≠n+1 ---> CalcPrimosCercanos(d(i),C(i));
              i:=i+1
od
```

CalcPrimosCercanos(d,C) es un subprograma cuyo texto está por escribirse (esta acción corresponde a la aplicación de una estrategia de abstracción); que recibe un entero d mayor que 1 como argumento, y deja en C el conjunto de primos más cercanos a éste.

Una vez calculado el texto de este subprograma el problema estará resuelto.

Antes de continuar aplicando la metodología a dicho subproblema conviene notar que la corrección del desarrollo que se tiene hasta ahora, es trivialmente verificable:

-P vale trivialmente después de la asignación i:=1
-P y no-(i≠n) (i.e. P y i=n) implican la postcondición R
-{P y i≠n} CalcPrimosCercanos(d(i),C(i)); i:=i+1 {P}

(Si CalcPrimosCercanos se comporta en la forma descrita arriba la terminación del problema es evidente.

Desarrollo del subprograma CalcPrimosCercanos

Este subproblema puede especificarse así:

Precondición Q_1: d es un entero mayor que 1
Postcondición R_1: C = {p: p es primo más cercano a d}

Se puede reducir la solución a dos casos:
d es primo y d no es primo (donde en el primer caso la solución es inmediata). Usando el comando de selección se obtiene

{Q_1}
if d es primo ---> C:={d}
[] d no es primo ---> S_1
fi
{R_1}

Aquí {d} es el conjunto cuyo único elemento es d y S_1 es un subprograma que debe cumplir la especificacion

$$\{Q_1 \text{ y d no es primo}\}\ S_1\ \{R_1\}$$

Para obtener S_1 se observa que la postcondicion R_1 se puede escribir así:

R_1: R_1' y C = {p: p es primo, y p=p_1 ó p=p_2} donde

R_1': -|---------|---------|---------|---> , p_1 o p_2 es primo y
0 p_1 d p_2
no existe ningún primo estrictamente entre p_1 y p_2.
La gráfica de arriba ilustra la situación relativa de p_1 y p_2 abreviando la formula "p_1 y p_2 son equidistantes a d y $p_1<p_2$".

Si se aplica ahora, la técnica de "dividir y conquistar" obteniendo primero a R_1' y luego a R_1 mediante la secuenciación $S_{11};S_{12}$ de tal forma qué S_{11} y S_{12} cumplan respectivamente

$$\{Q_1 \text{ y d no es primo}\}\ S_{11}\ \{R_1'\} \text{ y } \{R_1'\}\ S_{12}\ \{R_1\}$$

como $R_1 \Rightarrow R$, esto es suficiente para lograr el objetivo propuesto.

S_{12} es bastante sencillo y no amerita reflexiones adicionales:

```
if p1 es primo      ---> C:={p1}
[] p1 no es primo   ---> C:={}
fi;
if p2 es primo      ---> C:=C U {p2}
[] p2 no es primo   ---> skip
fi
```

{} denota el conjunto vacío, skip es un comando que no afecta ninguna variable (i.e. deja las cosas como estan).
Mucho más interesante es la construcción de S_{11}. Esta se logra mediante un ciclo iterativo cuyo predicado invariante consiste en un "logro parcial" de R_1:

P_1: -|----|----|-- > y no existe ningún primo estrictamente
p_1 d p_2
entre p_1 y p_2.

Observe que la inicialización $p_1:=d$; $p_2:=d$ cumple el invariante P_1, aunque se puede ser más sutil:

```
if d es par    ---> p1:=d-1; p2:=d+1
[] d es impar  ---> p1:=d-2; p2:=d+2
fi
```

La precondición de S_{11} garantiza que d no es primo. Además como P_1 y (p_1 es primo o p_2 es primo) implican R'_1, la guarda para el ciclo en construcción es: p_1 no es primo y p_2 no es primo; obteniendose

```
{Q1 y d no es primo}
if d es par    ---> p1:=d-1;p2:=d+1
[] d es impar  ---> p1:=d-2;p2:=d+2
fi;
{P1}
do ¬primo(p1) y ¬primo(p2) ---> "Preservar el invariante
                                 P1 incrementando la
                                 longitud del intervalo
                                 [p1,p2]"
od
```

Se puede pensar en primo(p) como una función lógica que habla sobre la verdad de la afirmación "p es primo".
La preservación del invariante para el cuerpo del ciclo se logra mediante el comando $p_1:=p_1-2$; $p_2:=p_2+2$. La corrección de este paso así como la terminación del ciclo estan garantizadas por propiedades elementales de los números primos.

En resumen se tiene

```
i:=1;
do i≠n+1 --->
    if primo(d(i))   ---> C:={d(i)}
    ▯ ¬primo(d(i)) ---> if d(i) es par   ---> p1:=d(i)-1;p2:=d(i)+1
                        ▯ d(i) es impar ---> p1:=d(i)-2;p2:=d(i)+2
                        fi;
                        do ¬primo(p1) y ¬primo --->
                                        p1:=p1-2;p2:=p2+2
                        od;
                        if primo(p1)  ---> C(i):={p1}
                        ▯ ¬primo(p1)  ---> C(i):={}
                        fi;
                        if primo(p2)  ---> C(i):=C(i) U {p2}
                        ▯ ¬primo(p2)  ---> skip
                        fi
    fi;
    i:=i+1
od
```

El desarrollo mostrado hasta este punto del programa es suficiente para dar una idea más o menos precisa de la propuesta matodologica que se pretendía exponer. Sin embargo el lector interesado podrá estudiar el diseño ulterior de la función decisora primo(p) en el apéndice al final de este artículo.

3. Conclusiones

El ejemplo anterior ha permitido ilustrar la correspondencia anunciada en la introducción, entre el lenguaje algorítmico usado, y las tres estrategias de solución de problemas ya mencionadas. Se ha visto también como la construcción del programa y la prueba de su corrección pueden "ir de la mano"; con la prueba de la corrección guiando la construcción.

Como subproducto de hacer de la programación una actividad "guiada por objetivos" mediante la especificación de cada etapa de la construcción del programa por medio de aserciones, se obtienen programas a la vez, claros, sucintos y elegantes.

Los métodos presentados en este artículo son de naturaleza deductiva, es decir van de lo global a lo particular. Para una propuesta metodológica basada en métodos inductivos ver Dromey [Dr85].

APENDICE

Se continuará aquí, la aplicación de la metodología de Dijkstra y Gries al problema en cuestión, diseñando una función que decida si un entero positivo es primo o no.
Tomando prestado un poco de la notación del lenguaje Pascal tendriamos:

```
function primo(d: nat): boolean;
{O2: d>0}
{R2:    (primo(d)=true  y d es primo)
     o (primo(d)=false y d no es primo)}
```

Para la elaboración de un plan de ataque al problema conviene estudiar la definición de número primo.

Un entero positivo d es primo si y solo si

(1) d≠1 y los únicos divisores positivos de d son 1 y d mismo.

Esta definición puede establecerse en forma positiva y acotada como

d≠1 y todo entero k que cumpla $2 \leq k < d$ no es un divisor de d.

Esta definición sugiere un estudio de casos y un ciclo iterativo para decidir si d es primo. Sin embargo se puede hilar más fino:

Un entero positivo d es primo si y solo si

(2) d=2 o (d es un impar mayor que 1 tal que todo número impar que cumpla $3 \leq k \leq [d]$ no es divisor de d)

([x] representa la parte entera (mayor entero menor que) del número x)

Se interpretará entonces la noción de primo de acuerdo a la definición (2)

Una reducción de casos obtendría el siguiente comando

```
{d>0}
if d=2                    ---> primo(d):= true
[] d>1 y d impar          ---> S3
[] d=1 o (d par y d>2)    ---> primo(d):= false
fi
{R2}
```

Donde S_3 es un subcomando que debe cumplir:

$\{d>1 \text{ y } d \text{ impar}\}\ S_3\ \{R_2\}$

S_3 se diseña mediante un ciclo iterativo cuyo predicado invariante se obtiene debilitando la aserción R_2 a P_2:

P_2: d>1, d impar y además i es un impar con $3 \leq i \leq [\ d]+2$ que cumple

(primo(d)=true y $\forall k$(k impar y $3 \leq k < i$ => k∤d))

o (primo(d)=false, $i \leq [\ d]$ y i∤d).

Es decir, i es una variable de trabajo que variará sobre los números impares desde 3 hasta máximo [d]+2 para la cual todos los impares estrictamente menores que ella no dividen a d y el valor de primo(d) es true; o el valor de p(d) es false, e i es un "testigo" de la no primalidad de d, por el hecho de dividirlo.

Por tanto P_2 y (primo(d)=false o i>[d]) implican R_2, es decir, (primo(d)=true y d es primo) o (primo(d)=false y d no es primo).

Además t=[d]-i es una cota superior para el número de iteraciones que aun queden por ejecutar en el siguiente ciclo:

```
{d>1 y d impar}
i:=3;
primo(d):=true;
{P2}
do i≤[ d] y primo(d)=true --->
            if i|d ---> primo(d):=false
            [] i∤d ---> i:=i+2
            fi
od
{R2}
```

BIBLIOGRAFIA

[Dj76] Dijkstra, E.W.: A Discipline of Programming. New Jersey: Prentice-Hall 1976

[Dr85] Dromey, R.G.: Program Development by Inductive Stepwise Refinement. Software Practice and Experience, Vol. 15(1), pp 1-28, Enero 1985

[G81] Gries, D.: The Science of Programming. Springer Verlag, N.Y., 1981

[H69] Hoare, C.A.R.: An Axiomatic basis for Computer Programming. CACM, 12, pp 576-80, 1969

[Hh79] Hehner, E.C.R.: **do** Considered **od**: A Contribution to the Programming Calculus. Acta Informatica 11, pp 287-304, 1979

[J75] Jackson, M.A.: Principles of Program Design. Academic Press, London, 1975

[W71] Wirth, N.: Program Development by Stepwise Refinement, CACM 14(4) pp. 221-227, Abril 1971

[Wl82] Welsh, J., Elder, J.: Introduction to Pascal. Prentice Hall, Inc. Englewood Cliffs, N.J. 1979

DISEÑO E IMPLEMENTACION DE TIPOS ABSTRACTOS DE DATOS

Una metodología basada en funciones recursivas

Rodrigo Cardoso R.
Universidad de los Andes
Bogotá - *Colombia*

Resumen: Se presenta una metodología intuitiva para el diseño y la implementación de tipos abstractos de datos (TADs), con ayuda de funciones recursivas. Un TAD se entiende como un conjunto de nombres o formas normales sobre los cuales se definen operaciones que reflejan la estructura de un universo intuitivamente bien conocido que se quiere modelar.

Cada TAD definido es por construcción suficientemente completo.

Se define una relación de equivalencia sobre el conjunto de formas normales que captura la semántica del TAD con base en operaciones que observan propiedades externas de los objetos que se modelan. Se definen conjuntos de estas operaciones que caracterizan la relación de equivalencia mencionada.

Se estudia la implementación de TADs con lenguajes de programación de tipo imperativo.

La teoría se ilustra con el ejemplo "clásico" del TAD Stack[X], que modela pilas de objetos de un tipo primitivo X. Al final se incluye un ejemplo más complejo, el TAD Cubo, que modela el funcionamiento del cubo de Rubik, para mostrar la práctica de la metodología en un caso no trivial.

Palabras Claves: tipo abstracto de datos (TAD), funciones recursivas, formas normales, reducción de términos, completitud suficiente, equivalencia semántica, implementación de TADs.

0. Motivación

Los tipos abstractos de datos (TADs) pueden ser una herramienta útil para la definición y manipulación de datos sobre los que operen programas de computador. Sin embargo, la teoría de TADs se ha desarrollado algo lejos de la práctica, de modo que se ha considerado interesante el estudio de problemas que nunca deberían aflorar en la realidad (v.gr. completitud suficiente) si el diseño se llevara a cabo siguiendo metodologías que justamente evitaran "por construcción" las grandes dificultades que se presentan en una teoría general de TADs.

En la práctica la preocupación debe ser proporcionar al observador (diseñador, programador) una metodología de trabajo que le permita representar simbólicamente un universo O (un conjunto de explícitos) que desea estudiar y operar, de modo que la representación refleje todas las características que el observador juzgue relevantes(1). Si para alcanzar este objetivo se usan TADs, es interesante entonces disponer de una metodología que guíe el diseño de un TAD T del cual O sea un modelo.

Lo que el observador considere importante de O debe ser capturado en T, y aunque es posible que algo más de lo "interesante" sea abstraído, el único modelo de T que interesa al observador es O. El TAD T no es otra cosa que un lenguaje formal para hablar de O, y cuando las cosas van bien, este lenguaje, que es una herramienta sintáctica, encierra todo lo que el observador sabe sobre O, es decir, su semántica.

Como el universo O debe ser "aprehensible", es natural pedir del observador que pueda describirlo de alguna manera efectiva. Por ejemplo, debería poder enumerarlo, o sea concebir una forma recursiva (intuitiva) para nombrar los elementos de O, que también puede pensarse como conocer una forma de construír todos los elementos de O. Esta hipótesis de constructividad intuitiva de O permite comenzar el diseño del TAD T definiendo (recursivamente) el conjunto subyacente o tipo de interés Y como un conjunto de nombres estándar o formas normales para los elementos de O, de modo que cada objeto tenga al menos un nombre.

Eventualmente el observador puede considerar transformaciones efectivas sobre objetos de O que no creen nuevos objetos. En el TAD T se tendrá como contraparte la existencia de funciones recursivas definidas sobre el conjunto de nombres Y. La misma idea se sigue para modelar en T el hecho de que el observador pueda analizar una característica de los objetos. La definición de estas funciones recursivas se hace en T por medio de axiomas que describen reglas de reducción; cuando se consiguen buenas definiciones la axiomatización es por construcción suficientemente

---

(1) Los símbolos en los que en últimas quiere representarse O son los objetos que provee y manipula un lenguaje de programación en el que se quiere simular la realidad.

completa. Es decir, cada vez que se quiere analizar alguna característica de un objeto, los axiomas bastan para calcular la función recursiva correspondiente a la característica, i.e. para determinar un resultado en el conjunto de posibles valores de la función.

La discusión anterior apoya un supuesto metodológico importante en lo que aquí se desarrollará. Se supone del observador una idea clara del universo O que le permita definir operaciones en el TAD T clasificadas así:

iniciales : nombres para objetos "simples" de O (no construíbles a partir de otros objetos de O ya conocidos).

constructoras : nombres de funciones constructoras de objetos de O, que reciben argumentos que son objetos de O ya entendidos.

simplificadoras: transformaciones en Y que reflejan transformaciones correspondientes en O.

analizadoras : funciones sobre Y que permiten observar características de los objetos de O.

Se llama selectora a una operación simplificadora o analizadora.

Los identificadores de estas funciones y sus funcionalidades conforman la signatura del TAD T. Asociada a esta hay un lenguaje formal cuyos términos pueden clasificarse según el símbolo de función más externo que lo constituye.

El conjunto de términos se notará F_Σ. Se llamará Y_Σ al conjunto de los términos cuyo símbolo de función más externo corresponde al tipo de interés. Y es subconjunto de Y_Σ, y resulta natural definir (recursivamente, apoyándose en las definiciones recursivas sobre Y de las operaciones simplificadoras) una función normalizadora que asigne a cada elemento de Y_Σ una forma normal. Una reducción similar puede hacerse con términos de tipos diferentes al de interés.

Después será sencillo extender recursivamente las definiciones de las diferentes operaciones a todo F_Σ, simplemente cambiando cada argumento en F_Σ por su forma normal y aplicando las definiciones ya conocidas.

La hipótesis de trabajo principal es que la estructura del conjunto de nombres Y refleja la estructura de construcción de O. Algo más es deseable: si el observador puede distinguir dos objetos o1 y o2 debería poder distinguir sus nombres y1, y2. La única manera de distinguir dos objetos es observando que tienen diferente alguna característica, y ésto se refleja en el TAD Y cuando para alguna función selectora el valor cambia si se sustituye y1 por y2 (y los demás argumentos no se modifican). La igualdad puede definirse entonces como ausencia de diferencias, y así se

justifica definir en Y una relación de equivalencia (semántica) tal que $y1 \equiv y2$ si y1 y y2 no son distinguibles mediante selectoras en el sentido anotado. Esta relación es naturalmente extendible al conjunto de términos del tipo de interés $Y_{\#}$.

La situación es verdaderamente satisfactoria cuando $.\equiv.$ captura en Y el conocimiento de O que tiene el observador. Algebraicamente: se quiere que $Y/\equiv$ y O sean "isomorfos", aunque tal noción debe dejarse también a la intuición, teniendo en cuenta el conocimiento apenas informal de O. Más precisamente, la definición del TAD T es satisfactoria cuando O es un álgebra (un modelo) inicial del TAD; cuando éste es el caso, dos elementos de Y son considerados diferentes si no se puede mostrar que son iguales (según $.\equiv.$).

Como notación, llámese T_{TDI} un TAD con tipo de interés TDI. Una implementación del TAD T_Y en otro TAD T_W define una subestructura del TAD T_W que es un álgebra inicial del TAD T_Y. Cuando la definición del TAD T_Y es satisfactoria en el sentido del párrafo anterior, la implementación define entonces una representación igualmente satisfactoria de O. Tal representación en el TAD T_W puede a su vez implementarse en otro TAD T_Z, y continuar esta cadena de representaciones hasta que se trate con objetos explícitos. En este artículo se considerarán explícitos los objetos primitivos que ofrezca el lenguaje de programación en el que se quiera representar el universo O. Este último paso en la cadena de representaciones no es esencialmente diferente de los anteriores si se exige un conocimiento formal de la clase de objetos que maneja el lenguaje de programación.

1. Estructuras computacionales. Tipos abstractos de datos

$E=\langle X,F\rangle$ es una estructura computacional (EC) si X es un conjunto de objetos explícitos y F un conjunto de operaciones sobre estos objetos. Típicamente E corresponde a los objetos provistos por un lenguaje de programación (v.gr. símbolos que representan números enteros, valores de verdad, etc.) y a las transformaciones sobre estos objetos predefinidas en el lenguaje (v.gr. suma de enteros, operaciones booleanas, etc.). Sobre X se supone definido un orden bien fundado $.<.$ (no hay cadenas infinitas descendentes).

$T=\langle X,F\rangle$ es un tipo abstracto de datos (TAD) cuando X es un conjunto de símbolos y F un conjunto de operaciones sobre estos símbolos y tal vez otros TADs. El conjunto X se llama el tipo de interés y se nota $tdi(T)= X$. Sobre X hay un orden bien fundado conocido $.<.$.

Un TAD es una EC o se construye a partir de TADs conocidos. A continuación se da una metodología para llevar a cabo una tal construcción de modo que el resultado sea una definición que tenga algunas propiedades muy deseables en la práctica (v.gr. completitud suficiente).

2. Metodología para construcción de TADs

La construcción de un TAD nuevo está motivada por el deseo de modelar un universo O entendido al menos intuitivamente. El conocimiento de O que tiene el observador puede entonces guiar la definición de T. Cada argumentación que se apoye en esta hipótesis de trabajo se señalará explícitamente en la siguiente descripción de la metodología de construcción de tipos como (HT)

2.1 Construcción de formas normales

(HT1) O es intuitivamente construíble. La construcción debe poderse hacer en forma recursiva, i.e. a partir de objetos simples o atómicos se construyen objetos más complejos. Los objetos simples y lo que se agrega a éstos para construír otros más complejos pueden describirse con símbolos tomados de TADs conocidos (2).

Sean T1,...,Tn TADs conocidos, con tdi(Tk)=Xk, k=1,...,n.
$N := \{1,...,n\}$, $N_0 := \{0,1,...,n\}$.

Por (HT1) se pueden definir dos conjuntos finitos de símbolos:

I : Conjunto de símbolos de operaciones iniciales. Con éstos se quieren nombrar los objetos simples de O. En general, los nombres pueden depender de los Xk's.

C : Conjunto de símbolos de operaciones constructoras. Con éstos se quieren construír los nombres de objetos complejos de O, a partir de nombres conocidos y algunos de los Xk's.

La aridad de los símbolos $f \in I \cup C$ se describe formalmente con dos funciones

$$m_X : I \cup C \dashrightarrow N^*,$$
$$m : I \cup C \dashrightarrow \underline{nat},$$

que indicarán respectivamente cuáles de los tipos primitivos Xk's y con qué multiplicidad el tipo de interés Y (que está por definirse!) participan en el dominio de definición del símbolo f.

Nótese que $m_X(f)$ es una sucesión finita de números j1,...,jk, $k \geq 0$. En este caso se notará $dom_X(f) := Xj1 \times ... \times Xjk$.

Cuando k=0 se nota $m_X(f) := \underline{nil}$, $dom_X(f) := \{\#\}$.

Se quiere además que m(i)=0 para $i \in I$, y $m(c)>0$ para $c \in C$.

Ahora se definen los conjuntos de nombres:

$$Y_0 := \{ i(x) \mid i \in I , x \in dom_X(i) \}$$

---

(2) Esta hipótesis se apoya en la Tesis de Church.

$Y_{j+1} := Y_j \cup \{c(x,\vartheta) \mid c \in C,\ x \in dom_X(c),\ \vartheta \in Y_j^{m(c)}\},\ j \geq 0.$

$Y := \bigcup_{j \geq 0} Y_j$: el conjunto de nombres estándar o <u>formas normales</u>.

Cuando para $i \in I$ se tiene que $dom_X(i)=\{\#\}$, se abrevia el nombre 'i(#)' mediante 'i'. Más generalmente, siempre que en una lista de parámetros simbólicos de un nombre deba aparecer '#', el símbolo se omite. El significado de esta convención es justamente denotar constantes con respecto a los Xk's, que se dan con frecuencia en especial para las operaciones iniciales.

Y se ordena según la complejidad de la construcción de sus elementos, y dentro de un mismo nivel de complejidad, lexicográficamente. El orden resultante es bien fundado.

2.1.1 Ejemplo: Modelaje de stacks o pilas

Como ejemplo clásico en cualquier presentación de TADs se incluye el estudio del TAD Stack[X], que abstrae pilas de objetos sacados de un tipo primitivo X (conocido, recursivamente enumerable, bien fundado). Las pilas son estructuras de almacenamiento "LIFO" ("last-in-first-out"), donde el último objeto guardado es el primero que puede retirarse.

Por simplicidad se usará X para denotar el tipo primitivo y su tipo de interés. En este caso: $X1=X$, $N=\{1\}$, $N_0=\{0,1\}$.

El stack más simple que se puede concebir es aquel que no tiene elementos. Es natural incluír un símbolo de operación que lo denote:

empty : símbolo de operacíon inicial. Denotará un stack vació de elementos de X.

A partir de un stack conocido se puede construír otro más complejo, agregando al primero un elemento más. La posibilidad de efectuar esta construcción justifica la inclusión de un símbolo de operación constructora:

push : símbolo de operación constructora. Operará sobre un stack s, y un elemento x.

Resumiendo: $I = \{empty\}$, $C = \{push\}$.

$m_X(empty)=\underline{nil}$, $m_X(push)=\langle 1 \rangle$,

$m(empty)=0$, $m(push)=1$,

Así: $dom_X(empty)=\{\#\}$, $dom_X(push)=X$.

$Stack_0 = \{empty\}$

$Stack_{j+1} = Stack_j \cup \{push(s,x) \mid s \in Stack_j,\ x \in X\}$

$Stack = \bigcup_{j \geq 0} Stack_j$: es el conjunto de formas normales para nombrar stacks.

El conjunto Stack se provee de un orden bien fundado, de acuerdo a la complejidad de la construcción y al orden del conjunto X.

2.2 Operaciones selectoras

(HT2) En O puede desearse transformar objetos en objetos. En T ésto se refleja en transformar los nombres correspondientes.

El poder analizar características de objetos de O se traduce en operaciones que analizan nombres y dan como resultado elementos de los Xk's. Esto supone que las partes de los objetos de O pueden nombrarse con símbolos de los Xk's.

Por (HT2) se definen conjuntos finitos de símbolos:

S : conjunto de símbolos de operaciones simplificadoras. Con éstos se quieren denotar las operaciones que transforman objetos de O en objetos de O sin crear nuevos objetos.

A : conjunto de símbolos de operaciones analizadoras. Con éstos se quieren denotar las operaciones que analizan características de los objetos de O.

S U A es el conjunto de operaciones selectoras.

F:= I U C U S U A es el conjunto de símbolos de operaciones o funciones de T.

La aridad de los símbolos $f \in S \cup A$ se describe extendiendo las funciones m_X y m:

$$m_X : F \dashrightarrow N^*,$$
$$m : F \dashrightarrow \underline{nat}.$$

Para $f \in S \cup A$ se quiere que $m(f)>0$.

El rango de cada $f \in F$ puede definirse mediante una función

$$r : F \dashrightarrow N_0,$$

tal que $r(f)=0$, si $f \in I \cup C \cup S$
$r(f)>0$, si $f \in A$.

La notación $dom_X(f)$ se extiende para $f \in S \cup A$, y se define
ran(f):= Y , si r(f)=0
Xk, si r(f)=k, k>0.

Asociada a cada $f \in S \cup A$ se define una función recursiva

$$f^* : dom_X(f) \times Y^{m(f)} \dashrightarrow ran(f)$$

mediante axiomas que se apoyan en el orden bien fundado de Y. La definición de cada f^* exige el conocimiento de la transformación correspondiente en los objetos de O (HT2).

2.2.1 Ejemplo: Modelaje de stacks o pilas (continuación)

En la manipulación de stacks se utilizan generalmente dos funciones selectoras, una simplificadora y una analizadora:

pop(s): denota el stack resultante de retirar del stack s el último elemento ingresado. Su acción sobre empty, el stack vacío, no se define.

top(s): denota el último elemento ingresado al stack s. También top(empty) se considera indefinido.

Entonces se definen los conjuntos: S = {pop}, A = {top}.

Y además:

$$m_X(pop)=\underline{nil},\ m_X(top)=\underline{nil},\ m(pop)=1,\ m(top)=1,$$

$$r(pop)=0,\ r(top)=1.$$

Entonces se necesita definir dos funciones recursivas

$$pop^*: Stack \dashrightarrow Stack$$
$$top^*: Stack \dashrightarrow X\ ,$$

cuya acción debe explicarse efectivamente, mediante axiomas que describan cómo operan sobre las formas normales de Stack.

Los siguientes axiomas definen recursivamente pop^* y top^*:

(S1) pop(empty) = $\underline{\perp}$
(S2) pop(push(s,x)) = s
(S3) top(empty) = $\underline{\perp}$
(S4) top(push(s,x)) = x .

$\underline{\perp}$ es una convención para denotar "indefinido" o "error". Ambos casos se consideran "anormales", y el uso de $\underline{\perp}$ como argumento de cualquier operación da como resultado $\underline{\perp}$.

2.3. Términos

En el presente contexto la cuádrupla (F,m_X,m,r) conforma la llamada <u>signatura</u> del tipo T. La signatura define naturalmente un conjunto F_Φ de <u>terminos</u> que constituyen un lenguaje formal para hablar del universo O.

El conjunto de términos se define inductivamente así:
(Sea $Ak := \{a \in A \mid r(a)=k\}$)

$$Xk_{[0]} := Xk,\ k=1,\ldots,n$$

$$Y_{[0]} := Y_0.$$

$$Xk_{[j+1]} := Xk_{[j]} \cup \{a(x,y) \mid a \in Ak,\ m_X(a)=q1\ldots qp,\ x \in Xq1_{[j]}x\ldots xXqp_{[j]},\ y \in Y_{[j]}^{m(a)}\}$$

para $k=1,\ldots,n$.

$Y_{[j+1]} := Y_{[j]} \cup \{f(x,\vec{y}) \mid f \in I \cup C \cup S,\ m_X(s)=q1\ldots qp,$
$x \in Xq1_{[j]}x\ldots xXqp_{[j]},\ \vec{y} \in Y_{[j]}^{m(s)}\}.$
para $j \geq 0$.

Ahora pueden definirse:

$Xk_\# := \bigcup_{j\geq 0} Xk_{[j]}$: el conjunto de los Xk-términos,

$Y_\# := \bigcup_{j\geq 0} Y_{[j]}$: el conjunto de los Y-términos,

$F_\# := \bigcup_{1\leq k\leq n} Xk_\# \cup Y_\#$: el conjunto de los términos.

2.3.1 Modelaje de stacks o pilas (continuación)

Los términos no son otra cosa que las fórmulas sintácticamente bien formadas de acuerdo a las funcionalidades de las operaciones. Con lo hasta aquí definido (formas normales, símbolos de operaciones,...), los X-términos son las expresiones bien formadas que tienen top como símbolo más externo, y los Stack-términos son las que tienen empty, push o pop como símbolo más externo.

Cada término debería nombrar un objeto; la completitud suficiente se refiere justamente al hecho de saber a qué objeto se refiere cada término que no es del tipo de interés. Por ejemplo, debería ser claro que el término

top(pop(push(push(empty,a),b)

se refiere al objeto primitivo a.

2.4 Reducción de términos

Se define una reducción

$$.' : F_\# \longrightarrow \bigcup_{1\leq k\leq n} X_k \cup Y$$

de la siguiente manera: (Notación: $(z_1,\ldots,z_u)'=(z_1',\ldots,z_u')$)

Si $t \in Xk_{[0]}$: $t' := t$, $k=1,\ldots,n$.

Si $t \in Y_{[0]}$: $t' := t$.

Si $t \in Xk_{[j+1]} \setminus Xk_{[j]}$, $t=a(x,\vec{y})$: $t' := a^{\#}(x',\vec{y}')$, $k=1,\ldots,n$.

Si $t \in Y_{[j+1]} \setminus Y_{[j]}$, $t=s(x,\vec{y})$: $t' := s^{\#}(x',\vec{y}')$

para $j \geq 0$.

Así, cada Xk-término reduce a un elemento de Xk, y cada Y-término a una forma normal en Y.

2.5 Definición del nuevo TAD T

A cada símbolo de operación $f \in F$ se puede asociar una función recursiva también denotada f:

$$f: dom_X(f) \times Y^{m(f)} \longrightarrow ran(f)$$

así:

Para $i \in I$: $i: dom_X(i) \longrightarrow Y$, $x \mapsto i(x)$

Para $c \in C$: $c: dom_X(c) \times Y^{m(c)} \longrightarrow Y$, $(x,y) \mapsto c(x,y)$

Para $f \in S \cup A$: $f := f^{\#}$.

El conjunto de funciones asociadas se nombra también F.

Nótese que Y es un conjunto de cadenas de símbolos de la forma i(x) o bien c(x,y). Las imágenes por las funciones $i \in I$ y $c \in C$ son entonces cadenas de símbolos. Para las $f \in S \cup A$ las imágenes son elementos de los Xk's (que a su vez pueden ser cadenas de símbolos, pero ésto es externo a la definición del nuevo TAD).

Ahora TAD T := <Y,F> es el nuevo TAD. Por construcción (HT1,HT2) O es un modelo intuitivo de T. Además T es suficientemente completo, porque en virtud de 2.4 todo término de la forma a(x,y) reduce recursivamente a un objeto primitivo.

2.5.1 Modelaje de stacks o pilas (continuación)

La siguiente notación resume la definición del TAD Stack[X]:

Tipo Stack[X]
Operaciones
*empty: ---> Stack
*push : Stack x X ---> Stack
pop : Stack ---> Stack
top : Stack ---> X
Axiomas
(S1) pop(empty) = $\perp$
(S2) pop(push(s,x)) = s
(S3) top(empty) = $\perp$
(S4) top(push(s,x)) = x
finTipo

Las operaciones iniciales y constructoras son marcadas con '*'. Esto permite conocer tácitamente las formas normales de los stacks. Es decir se puede enunciar y mostrar fácilmente el siguiente resultado:

LFN(Stack) (Lema de forma normal para Stack)
Todo $s \in$ Stack tiene exactamente una de las siguientes formas:
i) empty
ii) push(s1,x), con $s1 \in$ Stack, $x \in X$.

Solo hay axiomas para las operaciones selectoras. Debe haber un axioma por cada posible forma normal del tipo de interés, con lo cual se garantiza la buena definición de cada selectora. Las funciones son "totales", en el sentido de que se explican para todas las posibles instancias del dominio; cuando en algún argumento deben quedar indefinidas, ésto se hace explícito (S1,S3).

3. Equivalencia

Considérese el TAD T = <Y,F> de 2. La hipótesis de trabajo (HT1) no exige que cada objeto de O tenga un nombre único, y en la práctica es usual que un mismo objeto tenga varios nombres (v.gr. la notación {1,2,3} denota el mismo conjunto que {2,1,3}). Enseguida se quiere establecer cuándo dos nombres se refieren a un mismo objeto.

La siguiente notación es útil:
Para $\bar{y} = (y_1,\ldots,y_{m-1}) \in Y^{m-1}$, $y_0 \in Y$, $1 \le j < m$, se define:
$[\bar{y},y_0]_j := (y_1,\ldots,y_{j-1},y_0,y_{j+1},\ldots,y_{m-1}) \in Y^m$.

3.1 Definición
Sean y1, y2 $\in$ Y.

i) Sea f $\in$ S U A.
y1 $=_f$ y2 ssi y1 es f-equivalente a y2

ssi Para todo x $\in dom_X(f)$, $\bar{y} \in Y^{m(f)-1}$, $1 \le j < m(f)$:

$$f(x,[\bar{y},y1]_j) = f(x,[\bar{y},y2]_j).$$

ii) Sea K $\subseteq$ S U A.
y1 $=_K$ y2 ssi y1 es K-equivalente a y2

ssi Para toda f $\in$ K : y1 $=_f$ y2.

iii) y1 $\equiv$ y2 ssi y1 es (semánticamente) equivalente a y2

ssi y1 $=_{SUA}$ y2.

iv) K $\subseteq$ S U A es un caracterizador (de Y), si para todo par y1,y2 $\in$ Y:

$$y1 =_K y2 \Rightarrow y1 \equiv y2.$$

∎

Si y1, y2 nombran objetos o1,o2 respectivamente, y1 $=_f$ y2 si la selectora f no los distingue. Por (HT2) ésto significa que el observador tampoco distingue la característica correspondiente entre o1 y o2. Si además y1 $\equiv$ y2, con (HT2) se deduce que o1 y o2 son indistinguibles para el observador.

De ésta manera Y/$\equiv$ es intuitivamente isomorfo a O. Cuando en Y

$\equiv$ se considera como igualdad algebraica, O es un modelo inicial de T puesto que dos objetos son diferentes si no se puede probar que son iguales. La existencia de un caracterizador no trivial reduce el número de condiciones que deben verificarse para mostrar equivalencia.

3.1.1 Modelaje de stacks o pilas (continuación)

Para el TAD Stack[X] hay una operación simplificadora (pop), y una analizadora (top). Con estas dos funciones se define la equivalencia de dos stacks. Entonces:

s1 $\equiv$ s2 ssi pop(s1) $\equiv$ s2 and top(s1) = top(s2)
ssi s1 y s2 tienen el mismo elemento en el tope, y al retirarlo de ambos quedan stacks equivalentes.

El que esta sea una definición razonable hace creer que el TAD Stack[X] está bien definido, y que Stack/$\equiv$ es intuitivamente isomorfo al modelo que el diseñador pretende abstraer.

No hay caracterizadores no triviales en este caso. Dos stacks pueden tener iguales sus topes y diferir en el resto o viceversa. Sin embargo, ésto justifica la escogencia que se hizo de las funciones selectoras: cualquier otra definición de una selectora que no distinguiera alguna característica no considerada sería redundante para la determinación de la equivalencia de dos stacks.

4. Implementaciones

Sean $T_Y = \langle Y, F_Y \rangle$, $T_W = \langle W, F_W \rangle$ dos TADs construídos a partir de otros TADs conocidos $T1,\ldots,Tn$, con $tdi(Tk)=X_k$, $k=1,\ldots,n$.

4.1 Definición

Una implementación Imp = (ß,P) de T_Y en T_W consta de :

i) Una función recursiva parcial sobreyectiva
$$ß: W \dashrightarrow Y$$
llamada la representación (de W como Y).
Para $\hat{w}=(w1,\ldots,wp)$, $ß(\hat{w}) := (ß(w1),\ldots,ß(wp))$.

ii) Para cada $f \in F_Y$ una función recursiva
$$f_W: dom_X(f) \times W^{m(f)} \dashrightarrow Z,$$
donde $Z = ran(f)$, si $ran(f) \neq Y$
$= W$, si $ran(f)=Y$,

tal que: $f(x,ß(\hat{w})) = f_W(x,\hat{w})$, si $ran(f)\neq Y$
$= ß(f_W(x,\hat{w}))$, si $ran(f)=Y$.

$P := \{ f_W \mid f \in F_Y \}$

Cuando T_W es una estructura computacional hay un lenguaje de programación que la manipula. Las funciones f_W son calculadas por programas en este lenguaje.

Si el lenguaje de programación es de tipo funcional (v.gr. LISP) f_W es una función cuya definición está naturalmente guiada por la de f. Una tal implementación es útil como prototipo del TAD T_Y y para verificar experimentalmente la buena definición de T_Y.

Sin embargo, cuando se desea una implementación eficiente es más aconsejable un lenguaje de tipo imperativo (v.gr. PASCAL). Cada función es calculada con un procedimiento (3) pf_W cuya especificación está determinada por la definición de f en T_Y.

Más precisamente, si F_Y = I U C U S U A como en 2.:

Para i ∈ I, i_W se calcula con

```
proc pi_W (x: dom_X(i); var pw: W);
     {Pre : x=x_0}
     {Post: ß(pw) = i(x_0) }.
```

Para c ∈ C, c_W se calcula con

```
proc pc_W (x: dom_X(c); var ŵ: W^(m(c)-1), pw: W);
     {Pre : x=x_0, ŵ=ŵ_0, pw=pw_0 }
     {Post: ß(pw) = c(x_0,ß(ŵ_0),ß(pw_0)) }.
```

(Nótese el cálculo de c sobre uno de los parámetros.)

Para s ∈ S, s_W se calcula con

```
proc ps_W (x: dom_X(s); var ŵ: W^(m(s)-1), pw: W);
     {Pre : x=x_0, ŵ=ŵ_0, pw=pw_0 }
     {Post: ß(pw) = s(x_0,ß(ŵ_0),ß(pw_0)) }.
```

Para a ∈ A, a_W se calcula con

```
proc pa_W (x: dom_X(s); var ŵ: W^m(a), px: ran(a));
     {Pre : x=x_0, ŵ=ŵ_0 }
     {Post: px = a(x_0,ß(ŵ_0)) }.
```

---

(3) Eventualmente podrían usarse funciones, pero se adopta esta convención para facilitar la comparación con lenguajes como PASCAL, donde el resultado de una función solo puede ser de un tipo predefinido simple.
También se usarán las convenciones de PASCAL para notar el paso de parámetros.

4.1.1 Modelaje de stacks o pilas (continuación)

Para simplificar, supóngase X ≡ integer, el tipo primitivo de enteros PASCAL, y que se quiere implementar Stack[integer] en PASCAL.

Se necesita una función de representación, parcial, sobreyectiva:

ß: Estructuras de datos en PASCAL ----> Stack[integer],

y para cada operación f en Stack una realización PASCAL de la misma.

La definición de ß se deja usualmente a la experiencia de quien hace la implementación. En el caso de los stacks que se están modelando se puede observar, por ejemplo, que son estructuras no acotadas, y que su imagen PASCAL debería poder cambiar dinámicamente. Entonces es fácil pensar en listas simplemente encadenadas, donde cada elemento señala al anteriormente ingresado; el stack empty se representa con una lista vacía.

Una definición menos intuitiva de ß puede establecerse apoyándose en el LFN(Stack). Es decir, se busca definir, recursivamente, estructuras PASCAL para cada forma normal posible de Stack.

Cuando la experiencia es quien más ayuda a la definición de ß, las funciones selectoras son relativamente sencillas de programar, porque el observador visualiza en la estructura de datos que define los aspectos que le sirven para analizar el objeto. Por el contrario, cuando es el LFN la guía principal para el diseño de ß se fija además, implícitamente, la manera de realizar las funciones iniciales y constructoras.

Se puede ser tan formal como se quiera en la comprensión de ß; mientras más se conozca formalmente (menos intuitivamente), resultará más sencilla la realización de las operaciones del tipo. Sin embargo, es bueno encontrar un nivel de comprensión que permita justificar que la implementación es correcta sin caer en detalles técnicos exagerados.

Las operaciones del TAD Stack se realizan mediante procedimientos PASCAL tales que (pstack es el "type" PASCAL correspondiente a 'stack'):

```
procedure pempty (var p: pstack);
    {Pre : T }
    {Post: ß(p) = empty }

procedure ppush (var p: pstack; x: integer);
    {Pre : p = p0 and x = x0 }
    {Post: ß(p) = push(ß(p0),x0) }

procedure ppop (var p: pstack);
    {Pre : p = p0 }
    {Post: ß(p) = pop(ß(p0)) }
```

```
procedure ptop (var p: pstack; x: integer);
    {Pre : p = p0 }
    {Post: p = p0 and x = top(ß(p0)) }
```

4.2 La programación de una implementación en un lenguaje imperativo

Las especificaciones de procedimientos mencionadas en 4.1 son suficientes para programar una implementación del TAD T_Y en una EC $E_W = \langle W, F_W \rangle$. Es decir, si se tiene una metodología de desarrollo de programas a partir de especificaciones (cf. [Gri81], [Boh86]), lo que sigue entonces es un problema de programación.

Enfrentando un poco este problema se descubre que la principal dificultad es expresar las postcondiciones de una manera "cercana" al lenguaje de programación. O sea, puede ser necesario refrasear las postcondiciones de manera que se vea más claramente lo que se desea, para entonces aplicar las técnicas de desarrollo de programas que se estimen convenientes.

Típicamente cada postcondición involucra al menos una referencia a la función de representación ß. Esto sugiere la conveniencia de establecer con claridad esta función antes de atacar el problema de programar los procedimientos. Esta es la manera "clásica" de construír implementaciones: primero se diseñan las estructuras de datos que representarán los objetos, y después se programan las operaciones que manipulan estas estructuras.

Ahora bien, como ß es una función recursiva parcial sobreyectiva, lo usual es que esté definida apoyándose en el orden bien fundado de Y. Esto es, es de esperarse que se pueda decir, primero, cuál $w \in W$ es preimagen por ß de cada $i \in I$, y con esta información definir inductivamente la preimagen de cada y ($=c(x,\hat{y})$) en Y. De este modo, los procedimientos pi_W, $i \in I$, y pc_W, $c \in C$, resultan particularmente fáciles de implementar.

Las postcondiciones de los procedimientos pf_W, $f \in S \cup A$, se simplifican si se usan formas reducidas para x_0, $\hat{y}_0$ y pw_0, y se recuerda que en estos casos la función f coincide con la función f_W. Como también se está suponiendo una definición recursiva de estas funciones (apoyada en el orden bien fundado de Y), es entonces posible "acercar" la postcondición al lenguaje de programación.

El formalismo deja entrever otro orden para adelantar la implementación, menos "natural", pero igualmente válido. Supóngase que se tiene una idea de cómo son las especificaciones de las operaciones selectoras, aunque no se tenga claro cómo debería ser la implementación de las funciones iniciales y constructoras. Entonces se dispondría de un método para calcular las funciones $f \in S \cup A$, y por ende para decidir los predicados que involucren equivalencias semánticas. En particular, cada ß(pw) en las postcondiciones de los pi_W's y los pc_W's se puede ver como una incógnita, que una vez conocida establece una definición para ß, y da indicios de cómo programar estos procedimientos.

Esta última manera de llevar a cabo la implementación es especialmente viable cuando no hay operaciones simplificadoras. Cuando éste no es el caso, las postcondiciones de las operaciones simplificadoras incluyen equivalencias que a su vez dependen del conocimiento de las mismas operaciones simplificadoras. La aparente circularidad es inexistente si no hay operaciones simplificadoras.

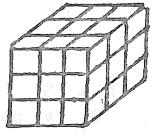
En cualquier caso el diseño se facilita si se dispone de un conjunto caracterizador no trivial que a su vez simplifique el manejo de la relación de equivalencia .=.

6. Un ejemplo más complejo: El TAD Cubo

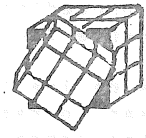
A continuación se incluye un ejemplo "no convencional" como ilustración de la metodología propuesta. Como se podrá observar, en la práctica no es necesario llevar a extremos el formalismo (v. gr. distinguir entre símbolos de operación y operaciones, cuidar la definibilidad de dominios y rangos, etc.), sin descuidar el rigor en el diseño y la implementación.

6.0 Conocimiento intuitivo

El cubo de Rubik es un rompecabezas tridimensional inventado por el profesor húngaro Ernõ Rubik. Se trata de un cubo constituído a su vez (aparentemente) por 27 elementos cúbicos que rotan en 3 direcciones por planos de 9 elementos. Algo como:



Posición normal



Rotación de la cara frontal

Las pequeñas caras de cada uno de los elementos cúbicos se llamarán "carillas", y están coloreadas de modo que hay una configuración del cubo que tiene cada cara (de 9 carillas) de un mismo color, y todas las caras tienen un color diferente.

El acertijo de Rubik consiste en, dada una configuración arbitraria del cubo, encontrar las rotaciones necesarias para llegar a una configuración en la que las 9 carillas de cada cara son de un mismo color.

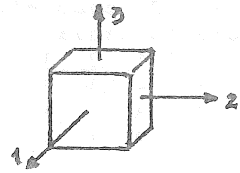
El TAD Cubo no pretende resolver el acertijo, sino modelar el funcionamiento del cubo de Rubik, i.e. las rotaciones y cambios de colores en las carillas. Sin embargo, el disponer de un lenguaje para hablar de estas características permite plantear for-

malmente el acertijo y buscar soluciones del mismo de una manera sistemática.

La metodología supone una idea intuitiva del objeto, que captura los aspectos y sus relaciones. Es quizás imposible separar estas ideas de lo que en el futuro será una representación del objeto, ya que al entenderlo debe existir alguna representación mental del mismo en la cabeza del observador.

En este caso es natural pensar el cubo como una caja de 3x3x3 enmarcada en un sistema de coordenadas cartesianas. Los elementos cúbicos se pueden identificar con su posición en estas coordenadas, y las carillas de cada elemento de acuerdo al eje en dirección al cual se mira el elemento.

Más concretamente, se toman como referencia tres ejes ortogonales fijos, numerados 1, 2, 3. Un objeto q ∈ Cubo se observa siempre en el origen de ese sistema de coordenadas, así:



Cada elemento cúbico tiene una posición

$$p = (p[1], p[2], p[3]),$$

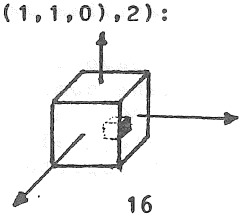
donde p[i] es un número en {-1,0,1}, i=1,2,3.

Cada elemento cúbico está compuesto de seis carillas. Para la descripción del cubo de Rubik se puede notar que un elemento cúbico tiene coloreadas a lo más tres de estas carillas, y más aún, a lo más una carilla coloreada en la dirección de cada eje del sistema de coordenadas. Entonces se puede simplificar un poco la descripción, haciendo que un elemento cúbico esté descrito por tres carillas, una para cada eje coordenado, algunas de las cuales están coloreadas y otras pueden tener un color neutro o indefinido.

Se puede decir, acercándose un poco más al cubo de Rubik, que un objeto q ∈ Cubo se compone de carillas identificables por una posición p -definida como arriba- y un eje que denote la dirección en que debe mirarse el elemento cúbico descrito por p.

De esta manera, una carilla es un par (p,e), donde p es una posición y e es un eje coordenado.

Por ejemplo, en el siguiente dibujo, la carilla sombreada se identifica con el par ((1,1,0),2):



Las siguientes notaciones son importantes:
Eje = {1,2,3} : el conjunto de ejes coordenados.
Dsp = {-1,0,1} : el conjunto de desplazamientos posibles con respecto a cada e ∈ Eje.
Posicion = Dsp^{Eje}
= {(p[1],p[2],p[3]) | p[i] ∈ Dsp, i=1,2,3 }
: el conjunto de posiciones de los elementos cúbicos de un cubo abstracto.
Pinta = {a,v,n,r,b,m}
: el conjunto de colores:
a: azul v: verde n: naranja
r: rojo b: blanco m: amarillo.

Además, para facilitar la descripción del tipo abstracto Cubo se utilizarán las siguientes funciones:

comp: Eje x Eje ---> Eje
donde comp(e1,e2) es el eje diferente a e1 y e2
(si e1=e2, comp(e1,e2) no se define!).
i.e. comp tiene la siguiente tabla:

	1	2	3
1	-	3	2
2	3	-	1
3	2	1	-

r: Posicion x Eje ---> Posicion
donde r(p,e) es una rotación de la posición p, con eje e, 90° en el sentido de las manecillas del reloj.
i.e. r(p,e) = if e=1 --> (p[1], p[3],-p[2])
| e=2 --> (-p[3], p[2], p[1])
| e=3 --> (-p[2], p[1], p[3])
fi.

Obsérvese que Eje , Posicion, Pinta, comp, r son nociones correspondientes a tipos primitivos que en el momento no interesa estructurar. Basta considerarlos conocidos e implementables.

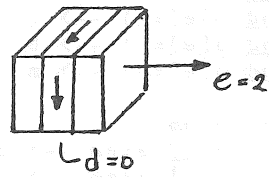
6.1 Construcción de formas normales

Cualquier cubo q ∈ Cubo se construirá a partir de un cubo inicial estándar, mediante rotaciones. Las cosas se simplifican si el cubo inicial estándar tiene los colores dispuestos según "la solución" del acertijo. Esto da lugar a una operación inicial, llamada inic, y a una constructora , llamada rote, entendidas de la siguiente manera:

6.1.1 inic : denota el cubo inicial estándar, que tiene un mismo color en todas las carillas de cada cara.

6.1.2 rote : es una función que, dado un cubo q, un eje e, y un desplazamiento d, tiene como valor el cubo resultado de rotar con eje e, 90° en el sentido contrario de las manecillas del reloj, el plano en q que tiene un desplazamiento d con respecto al eje e.

Por ejemplo, rote(q,2,0) denota un cubo como q en el que el plano central con respecto al eje 2 se ha rotado 90^{o} en el sentido contrario de las manecillas del reloj:



6.1.3 Como siempre, después de conocer las operaciones iniciales y constructoras es posible enunciar y demostrar el lema de forma normal del tipo de interés:

LFN(Cubo) (Lema de forma normal de Cubo)
Todo q ∈ Cubo es de una de las siguientes formas:
i) inic
ii) rote(q1,e,d), con q1 ∈ Cubo, e ∈ Eje, d ∈ Dsp.

6.2 Operaciones selectoras

Hay una función color, para denotar el color de cada carilla de un cubo.

Adelantándose un poco a lo que será la relación de equivalencia semántica derivada de las operaciones selectoras, es natural esperar que dos cubos sean equivalentes cuando sus carillas correspondientes tengan el mismo color. Así se justifica el hecho de que color sea la única función selectora en el TAD.

Es posible que una implementación requiera (para su eficiencia) un lenguaje más rico, v.gr. considerar rotaciones en el sentido de las manecillas del reloj. Estas pueden introducirse fácilmente una vez implementadas las operaciones hasta aquí mencionadas.

6.3 La definición del TAD Cubo

La siguiente definición resume el diseño del TAD Cubo:

Tipo: Cubo

Operaciones:

```
*inic :                              ---> Cubo
*rote : Cubo x   Eje    x Dsp ---> Cubo
 color: Cubo x Posicion x Eje ---> Pinta
```

```
Axiomas:
(C1) color(inic,p,e)
          = if e=1 and p[e]= 1 --> a
             ▯ e=1 and p[e]=-1 --> v
             ▯ e=2 and p[e]= 1 --> n
             ▯ e=2 and p[e]=-1 --> r
             ▯ e=3 and p[e]= 1 --> b
             ▯ e=3 and p[e]=-1 --> m
            fi
(C2) color(rote(q,e,c),p,em)
       = if p[e] = c --> if e=em --> color(q,r(p,e),em)
                         ▯ e≠em --> color(q,r(p,e),comp(e,em))
                        fi
         ▯ p[e] ≠ c --> color(q,p,em)
        fi
finTipo
```

Los axiomas son más complicados que los correspondientes al TAD Stack[X]. En el lado izquierdo se permiten expresiones if...fi y recursión.

6.4 Equivalencia

q1 ≡ q2 ssi Para toda carilla (p,e):
color(q1,p,e) = color(q2,p,e).

6.5 Implementación

Se esboza aquí una implementación PASCAL del TAD Cubo.

El primer paso es establecer la representación para los conjuntos y operaciones considerados primitivos al entender intuitivamente los cubos de Rubik.

Por ejemplo:

```
type Eje = 1..3;
     Dsp = -1..1;
     Posicion = array [Eje] of Dsp;
     Pinta = (a,v,n,r,b,m,indef)
      /* 6 colores definidos: azul,verde,...,amarillo.
         indef: color indefinido, para carillas interiores */
```

e implementar comp y r de acuerdo a lo definido en 2.0. Para r se necesita un procedimiento, puesto que el resultado no es un "type" simple, algo como

```
procedure pr (var pos_inicial,e,pos_final);
                    . . .
```

El segundo paso es definir la función de representación ß. Ya con las definiciones anteriores es natural escoger como representación PASCAL para un cubo una matriz de 3x3x3 de elementos cúbicos; cada elemento cúbico se representa a su vez con un vector de

3 puestos, cada uno de los cuales representa el color de la carilla visible del elememto en la dirección de un eje, si ésta lo es. Si no hay carilla visible se usa el color 'indef'.

Más exactamente, se incluye una definición
type pcubo = array [Dsp,Dsp,Dsp] of array[Eje] of Pinta

(sería deseable poder decir en PASCAL
array [Posicion] of array[Eje] of Pinta,
pero ésto no es sintácticamente posible).

Aquí se da una situación interesante: la operación selectora 'color' es más fácil de implementar que las constructoras 'inic' y 'rote', porque color(ß(pq),p,e) coincide con pq[d1,d2,d3][e], cuando p es la posición descrita por el vector de desplazamientos [d1,d2,d3]. O sea, el cálculo del color de una carilla es directamente consultable en la estructura que representa el cubo. De hecho, es inclusive ineficiente escribir código para esta función.

La operación 'inic'no es otra cosa que una inicialización de la estructura que representa el cubo que pasa como parámetro. Concretando:

```
procedure pinic (var pq: pcubo)
    {Pre : T }
    {Post: ß(pq) = inic }
```

Ahora, la postcondición significa que al terminar pinic, la variable pq, transformada por la función de representación ß, es equivalente a inic. Es decir:

"Para toda posición p, y para todo eje e:
color(ß(pq),p,e) = color(inic,p,e). "

O sea:

"Para toda posición [d1,d2,d3], y para todo eje e:
pq[d1,d2,d3][e] = color(inic,[d1,d2,d3],e). "

Ahora bien, el axioma (C1) de la definición del TAD Cubo se refiere justamente al color del cubo inic. Así, para cumplir con la postcondición de pinic es suficiente observar este axioma y efectuar en la estructura de datos lo anotado allí.

Una discusión análoga se puede adelantar para implementar el procedimiento correspondiente a la operación constructora 'rote'. Se necesita algo del estilo

```
procedure prote (var pq: pcubo; e: Eje; d: Dsp);
    {Pre : pq = pq0  and e = e0  and d = d0 }
    {Post: ß(pq) = rote(ß(pq0),e0,d0) }.
```

Y la postcondición se refrasea así
"Para toda posición [d1,d2,d3], y para todo eje e:
pq[d1,d2,d3][e] = color(rote(ß(pq0),e0,d0),[d1,d2,d3],e)."

El axioma (C2) de la definición del TAD Cubo sirve para desarrollar el código PASCAL para prote.

7. Conclusiones

Las ideas expuestas configuran la base teórica para una metodología de diseño e implementación de TADs.

El posible éxito de la aplicación de la metodología depende fundamentalmente del buen conocimiento intuitivo del universo que se pretende modelar. Una tal exigencia no parece muy restrictiva en la práctica; por el contrario, se puede afirmar que si no se puede hablar con rigor de un universo, mal puede pretenderse intentar un modelo formal que lo describa.

Los TADs diseñados son por construcción suficientemente completos, de modo que la obtención de esta importante propiedad deja de ser un problema. La dificultad se translada al hecho de poder dar axiomas que definan funciones recursivas sobre dominios dotados de órdenes bien fundados, y este problema es más fácil de atacar, sobre todo si, como se supone, se cuenta con una buena visión intuitiva de los objetos que se quieren modelar.

Es importante recalcar que el conjunto subyacente de un TAD así diseñado está constituído por cadenas de símbolos que vienen a ser nombres estándar o formas normales para los objetos del universo modelado. Aunque parezca artificial (y aún lejano de la práctica), esta manera de considerar los TADs es precisamente un indicio para establecer qué tan bueno es el conocimiento intuitivo que del universo a modelar tiene el diseñador: los elementos de un conjunto "bien conocido" deben poderse bautizar de alguna manera sistemática (un matemático diría "enumerar recursivamente").

La implementación de un TAD en una EC imperativa merece, por supuesto, un estudio mucho más profundo que el aquí anotado. Puede resultar especialmente interesante el caso en que las implementaciones de las operaciones iniciales y constructoras se apoyan en las de las funciones selectoras. Un mejor conocimiento de la relación de equivalencia semántica (por ejemplo buscando un conjunto caracterizador pequeño, y si se puede minimal) facilita en cualquier caso la verificación de la corrección de una implementación del TAD.

****

BIBLIOGRAFIA

[Bau82] Bauer F.L., Wössner H., Algorithmic Language and Program Development, Springer Verlag, 1982.

[Boh86] Bohórquez, J., "Cálculo de programas: una metodología precisa para el diseño de programas de computador", Memo de Investigación No.6, Universidad de los Andes - CIFI, Bogotá, 1986.

[Car86] Cardoso, R., "Diseño e implementación de tipos abstractos de datos", Memo de Investigación No. 9, Universidad de los Andes-CIFI, Bogotá, 1986.

[Car86] Cardoso, R., "Prácticas en tipos abstractos de datos", Memo de investigación (por aparecer), Universidad de los Andes-CIFI, Bogotá, 1986.

[Gri81] Gries, D., The Science of Programming, Springer Verlag, 1981.

[Gut77] Guttag, J.V., Horowitz E., Musser D.R., "The Design of Data Type Specifications", en Current Trends in Programming Methodology, R.T. Yeh (edt.), Prentice-Hall, 1978.

[Kuc85] Kucherov, G.A., "An Algorithm to recognize Sufficient Completeness of Algebraic Specifications on an Abstract Data Type", traducido de Programmirovanie, No. 4, pp.3-12, Julio-Agosto 1984.

[Pes83] Pessoa, E.P., Veloso, P.A., Introdução à especificação e implementação de tipos abstratos de dados, Monografias em Ciencia da Computação, No. 20, Pontifícia Universidade Católica do Rio de Janeiro, 1983.

****

ADA COMO LENGUAJE DE ESPECIFICACION

Juan Grompone
Montevideo - *Uruguay*

1. INTRODUCCION

Este trabajo se basa en la experiencia recogida en la especificacion, y posterior implementacion, de una Central Telex concebida como un conjunto distribuido de procesadores. Este proyecto, realizado durante 1985, era la tercera experiencia de diseño de un equipo de este uso, de modo que el tema era conocido y tambien eran conocidas las dificultades de ingenieria de Software que se encontraban en un proyecto de esta naturaleza.

El diseño original del Sistema era de 1978-1980 [1]. Debido a la falta de experiencia en proyectos de esta dificultad, la metodologia de trabajo fue mala. En 1981-1982 [2] se realizo un rediseño importante y se mejoraron muchos aspectos de la ingenieria de Software. En ambas oportunidadades se habian empleado tecnicas convencionales para el diseño y la documentacion. Las definiciones del proyecto eran informales y se reunian en "carpetas de diseño" que contenian anotaciones manuscritas, diagramas con "rombos y rectangulos", esquemas, fragmentos de programas y toda suerte de material no homogeneo.

La segunda experiencia mostro que eran necesarios metodos mas formales de trabajo. Las principales ideas de rediseño fueron publicados en [3]. De aqui surgio la conviccion que los nuevos proyectos emplearian una base teorica mas importante.

En la presente experiencia se decidio desde un principio elaborar un documento formal, conocido como **Manual de Especificacion**, en el que se establecieron reglas precisas. Estas reglas se pueden resumir en la frase: **toda la especificacion se realizaria con programas ADA** [4,5].

El Manual de Especificacion llego a la version 11 a lo largo de sus sucesivos refinamientos. En los cuatro primeros capitulos se especificaba el Hardware del Sistema, con especial enfasis en el diseño de la programacion. En los cinco siguientes se especificaba la programacion de Tiempo Real. En los tres ultimos se especificaba el Lenguaje Hombre Maquina que usaban las Consolas del Sistema.

En este trabajo se analizan las tecnicas y los resultados de esta experiencia, considerada altamente positiva en todos sus aspectos.

2. VENTAJAS DE ESPECIFICAR EN UN LENGUAJE CON COMPILADOR

Si bien muchas veces se ha sostenido la importancia de emplear un lenguaje formal de especificacion para los problemas de computacion [7], en este trabajo deseamos poner enfasis en el empleo de un lenguaje de especificacion **compilable**, de uso general, frente a especificaciones verbales, diagramas o lenguajes creados por el diseñador.

En forma breve, una especificacion formal, realizada en un lenguaje de alto nivel (es decir, alto nivel respecto al nivel de implementacion) permite **abstraer** los elementos esenciales y separar asi **diseño** de **implementacion**. Es un punto esencial del problema la eleccion del lenguaje de especificacion de manera de lograr que esta separacion sea efectiva.

Un lenguaje formal que posee un compilador asegura **la coherencia y la completitud** de la especificacion: al compilar la especificacion se detectan todos los errores formales. Este tipo de verificacion no puede ser realizada con un lenguaje creado para una aplicacion y, la mayoria de las veces, poco formal (en el caso de ADA se disponia solamente de un compilador que implementaba parcialmente el lenguaje, pero de todos modos presto gran utilidad en el proyecto [6]).

Con caracter general, las especificaciones formales permiten, ademas, algunas de las siguientes ventajas:

- definir en forma precisa algoritmos, estructuras de datos, nombres simbolicos y las acciones sobre estos objetos.

- reemplazar diagramas o descripciones verbales por objetos mas precisos.

- reemplazar las descripciones funcionales -por ejemplo del Hardware- por descripciones mas cercanas al implementador.

Estas caracteristicas justifican el empleo de lenguajes formales en general, pero la propuesta de ADA requiere argumentos adicionales. Las razones para elegir ADA como lenguaje de especificacion se pueden resumir en los siguientes puntos:

- el objetivo principal del lenguaje es la confiabilidad y no la facilidad de uso: esta mas cerca de un lenguaje de especificacion que de implementacion.

- es un lenguaje "grande", con muchas primitivas y opciones lo cual da una gran flexibilidad para un

especificador. Vale la pena notar que esta caracteristica es frecuentemente señalada como un inconveniente de ADA porque se lo contempla desde el punto de vista del fabricante de compiladores o del hardware a utilizar [8,9].

- es un lenguaje fuertemente orientado a tipos, a la organizacion modular y es muy exigente en sus aspectos formales.

- deriva de PASCAL y de alli que no se requiera un entrenamiento especial para comprender la especificacion.

- posee primitivas de tiempo real. Con el desarrollo de compiladores y aumento de potencia de los microprocesadores se convertira en un lenguaje de mucha difusion en esta area.

- puede suponerse que posee mucho futuro como lenguaje -mas que lenguajes ocasionales definidos por implementadores- puesto que es un lenguaje moderno, impulsado por el Departamento de Defensa de los Estados Unidos de Norte America.

En lo que sigue de este trabajo se presentan ejemplos concretos que ilustran estas afirmaciones.

3. EJEMPLOS DE ESPECIFICACION DE DATOS

Si bien se suele trabajar con descripciones bastante precisas de las estructuras de datos, muchos lenguajes de implementacion permiten trabajar libremente con los tipos, ya sea porque no imponen restricciones como ocurre con el Assembler, ya sea porque no hay estructuras de tipos abiertas y se deben emplear solamente enteros, caracteres, etc. Esta es una de las fuentes de errores mas importantes en el diseño y la implementacion de sistemas [10].

En el ejemplo que consideramos se manejan ciertos tipos de octetos, los caracteres Baudot (es decir los caracteres del telex), en forma permanente. La falta de precision en este punto es una fuente de errores.

Tomemos la descripcion de los caracteres que intercambian los procesadores de la Central Telex, segun las ideas de 1978:

"La comunicacion entre Procesadores Centrales y Procesadores Perifericos intercambia octetos que pueden ser caracteres Baudot, ordenes o mensajes. Los caracteres Baudot poseen la distribucion de bits:

0 xxxxx 11

El bit mas significativo es nulo. Por el contrario, las ordenes que los Procesadores Centrales envian a los Perifericos poseen siempre el bit mas significativo en 1. Los mensajes poseen ambos formatos."

En esta descripcion solamente se insiste en la estructura de bits de los caracteres Baudot, ordenes o mensajes. Este es un detalle de implementacion, casi sin importancia. No es trivial, en cambio, el hecho que los caracteres Baudot, ordenes o mensajes son muy diferentes de los contadores de tiempos o de la cantidad de caracteres de un texto, de los punteros y de los demas objetos. En la descripcion de 1985, en cambio, el enfasis se pone en la estructura de tipos:

```
package CONSTANTES is

-- caracteres BAUDOT

type BAUDOT is(

     NUL,     -- contenido 00H

     A,       -- contenido 63H

     B,       -- contenido 4FH

     --

     Z,       -- contenido 47H

     -- caracteres de control

     LE,       -- contenido 7FH: caracter letras

     CF,       -- contenido 6FH: caracter cifras

     SP,       -- contenido 13H: espacio

     --

     -- ordenes hacia el periferico

     OTR,      -- contenido 80H: comenzar comunicacion

     OLL,      -- contenido 81H: llamar

     OCX,      -- contenido 82H: conexion

     --
```

```
);
-- otras definiciones del paquete
end CONSTANTES;
```

La estructura efectiva de bits se encuentra al nivel de un comentario: se ha realizado la **abstraccion**. Notar que existe **sobrecarga** en la definicion de los caracteres **NUL**, **SP**, **A**, **B**, etc. y que este hecho es expresivo porque representan los mismos caracteres graficos que en el alfabeto ASCII.

Tomemos las descripciones de areas segun las ideas de 1978 y 1981. En aquel entonces no existia diferenciacion de tipos y la especificacion de areas era puramente verbal, solamente se ocupaba de las reservas de memoria (tal como le interesa al Assembler). Presentaba este aspecto:

```
nombre  largo   descripcion

N         -     numero de lineas de la Central: 256
...
ADE      256    area de entrada de datos de los perifericos
ADS      256    area de salida de datos hacia perifericos
...
CDOC     256    cola de mensajes de documentacion
CMENS    256    cola de mensajes a consola
...
LDE      514    lista de estados de lineas
...
FTEMP      1    bandera de temperatura
...
TIMER      2    contador BCD de decimas de minuto
...
XD         1    puntero de salida de CDOC
```

En lo que sigue se presenta la version de la misma estructura de datos, pero ahora bajo la forma de un paquete ADA:

```
package DATOS is
     --
   N    : constant integer := 256;     -- numero de lineas
     --
   ADE  : array(0..N-1) of BAUDOT;     -- area de entrada
   ADS  : array(0..N-1) of BAUDOT;     -- area de salida
     --
   CDOC : array(0..255) of character; -- cola de documentacion
   CMENS: array(0..255) of BAUDOT;     -- cola de mensajes
     --
   FTEMP: boolean := FALSE;       -- alarma de temperatura
     --
   LDE  : array(0..N) of ESTADO_DE_LINEA -- lista de estados
        := (0..N-1 => ELB, N => FIN );
     --
   TIMER: array(0..3) of BCD;      -- decimas de minuto
     --
   XD   : integer range 0..255 := 0;  -- puntero de CDOC
     --
end DATOS;
```

Como puede apreciarse, el paquete DATOS no solamente diferencia claramente nombres y tamaños, tal como se requiere para la implementacion en Assembler, sino que tambien identifica tipos y valor inicial en los casos que interese. Resulta muy expresivo conocer la diferencia de contenido de las colas CMENS y CDOC o introducir los tipos BCD (decimal

empaquetado) y ESTADO_DE_LINEA (definido en la Seccion 4).

Si bien el problema de la especificacion de las estructuras de datos es de caracter general, se convierte en un problema agudo cuando la implementacion se realiza con lenguajes que permiten gran libertad. Esto ocurre en el lenguaje C y en los Assembler y un caso tipico es el manejo de punteros. En estos lenguajes se permite una libertad que puede finalizar en una estructuracion deficiente, en muchos errores y en un mantenimiento muy dificil. La especificacion en ADA obliga a un manejo muy cauteloso de los punteros debido a las restricciones fuertes que impone el lenguaje.

4. EJEMPLOS DE ESPECIFICACION DE ALGORITMOS

El empleo de un lenguaje del alto nivel permite definir con mucha precision los algoritmos a emplear. Presentamos aqui algunos casos que son particularmente interesantes en ADA.

Una aplicacion tipica de ADA, como lenguaje con primitivas orientadas al Hardware, consiste en la descripcion de la informacion de Hardware. Los manuales que describen las piezas del equipo suelen estar escritos por ingenieros electronicos y destinados -como consecuencia- a ingenieros electronicos. Para que los ingenieros de formacion en computacion puedan comprender estos manuales es necesario traducirlos. ADA ofrece una manera muy natural de hacerlo. Este es un ejemplo de uso de una memoria no volatil que posea una de las maquinas usadas, donde la descripcion "tipo Hardware" se reemplaza por esta descripcion "tipo Software":

```
AREA_NO_VOLATIL: array(0..1023) of integer;

for AREA_NO_VOLATIL use 16#8000#;

function LEER(dir: integer) return integer is

DATO: integer;

begin

     out(16#20#, 1);        -- seleccionar la memoria

     DATO:=AREA_NO_VOLATIL(dir);

     out(16#20#, 0);        -- deseleccionarla

     return DATO;

end LEER;
```

```
procedure ESCRIBIR (dir, dato: integer) is

--
```

En este caso se usa fuertemente el procedimiento out que reemplaza una accion de entrada/salida del microprocesador. La mayoria de los programadores pueden comprender perfectamente como se maneja esta area no volatil pero no pueden comprender el Manual de Hardware. Es interesante observar que se emplea aqui la clausula de representacion de ADA: pocos lenguajes permitirian este nivel de especificacion.

Un tipo de algoritmo que se presenta frecuentemente es la realizacion de una maquina de estados. Presentaremos un caso tipico del proyecto de Central Telex: cada linea recorre una secuencia de estados. La descripcion de los estados de cada linea se realiza mediante un tipo definido por enumeracion. Una version simplificada se especifica por el texto ADA que sigue:

```
type ESTADO_DE_LINEA is(

    EAL      ,-- linea a liberar

    ECA      ,-- linea en comunicacion

    ELB      ,-- linea libre

    --

    FIN      -- estado de salida

);
```

Con esta definicion de estados, la especificacion de las transiciones -problema tipico de especificacion- se convierte en un algoritmo tipo case y en un problema abstracto, independiente de la implementacion. En el caso de la Central Telex el numero de estados es superior a 120. El diagrama de estados simplificado de la version 1978 ocupaba una hoja de 70 x 90 cm y era sumamente dificil de seguir y actualizar. En 1981 no se intento realizar nuevamente el diagrama. Por otra parte, un diagrama de estados solamente permite indicar algunos elementos de las transiciones pero no toda la actividad propia de cada estado. En el ejemplo ADA que sigue se ilustra la tecnica y se muestra, en forma simplificada, la manera de especificar la actividad y la transicion de estados.

```
procedure ATENCION_DE_LINEAS
          (ADE,ADS: BAUDOT, ESTADO: ESTADO_DE_LINEA) is
begin
case ESTADO is
     when EAL    =>
          case ADE is
               when A..Z =>
                    -- actividad
                    ESTADO:=ELB;
               when others =>
                    raise ERROR_DE_SISTEMA;
          end case;
     when ECA    =>     -- actividad
     when ELB    =>     -- actividad
     --
     when FIN    =>     -- actividad
end case;
end ATENCION_DE_LINEAS;
```

En este segmento, ademas de una especificacion de una secuencia de estados, que podria escribirse en muchos otros lenguajes, aparece una excepcion. Este es uno de los puntos mas importantes de ADA como lenguaje de especificacion. Contrariamente a lo que se ha sostenido [8], detectar y manejar en forma abstracta las excepciones es un punto fuerte de un diseño y una garantia de buena implementacion.

Es interesante estudiar otro ejemplo, una secuencia de comunicacion. En la tabla que sigue aparece toda la secuencia de comunicacion, de milisegundo en milisegundo, vista por los Procesadores Centrales y por los Procesadores Perifericos, segun las especificaciones de 1978 y 1981:

	CENTRALES		PERIFERICOS		
NINTE	lee	escribe	lee	escribe	CONTA
1			0	MENSAJE	62
	MENSAJE	80H			
2			80H	MENSAJE	-17
	MENSAJE	LINEA=1			
3			LINEA=1	LINEA=1	-16
	LINEA=1	LINEA=2			
. . . . .					
18			LINEA=16	LINEA=16	-1
	LINEA=16	0			

Consideramos que esta descripcion solamente puede ser comprendida si se la acompaña de una larga explicacion verbal. En el documento se decia, por ejemplo, que se empleaban lineas diferentes para la actividad de Perifericos y Centrales porque las acciones de los Perifericos ocurrian antes que las acciones de las Centrales. Aun asi pueden quedar muchas dudas. A titulo de ejemplo, al redactar la especificacion precisa, en ADA, se encontro un error en la tabla que no habia sido advertido en mas de 6 años.

Esta misma descripcion se puede realizar mediante un procedimiento ADA muy simple y que cualquier implementador puede comprender. Consideremos el caso del Procesador Central (el caso Periferico es similar):

```
procedure COMUNICACION (NINTE: integer) is
XIN:  integer:=0;        -- numero de linea en entrada
XOUT: integer:=0;        -- numero de linea en salida
procedure PERIFERICO_OUT (B: BAUDOT) is separate;
function  PERIFERICO_IN  return BAUDOT is separate;
begin
delay TIEMPO_DE_SEGURIDAD;
case NINTE is
```

```
            when 1 =>
                 MENSAJE_1 := PERIFERICO_IN;
                 PERIFERICO_OUT (16#80#);
            when 2 =>
                 MENSAJE_2 := PERIFERICO_IN;
                 PERIFERICO_OUT ( ADS(XOUT) );
                 XOUT:=XOUT+1;
            when 3..17 =>
                 ADE(XIN) := PERIFERICO_IN;
                 PERIFERICO_OUT ( ADS(XOUT) );
                 XIN:=XIN+1;
                 XOUT:=XOUT+1;
            when 18 =>
                 ADE(XIN) := PERIFERICO_IN;
                 PERIFERICO_OUT ( NUL );
                 XIN:=XIN+1;
       end case;
       end COMUNICACION;
```

En este ejemplo se trabaja con la funcion PERIFERICO_IN y el procedimiento PERIFERICO_OUT. Por lo demas, el programa ADA resulta mas preciso que la tabla: se señala que existe una demora de tiempos (TIEMPO_DE_SEGURIDAD), se diferencian los dos mensajes del Periferico, se indica cual es el area de entrada y cual es la de salida (ADE y ADS respectivamente) y se detecta un error de especificacion. En efecto, la constante 16#80# no es del tipo BAUDOT, debe emplearse, en lugar del valor, el caracter BAUDOT correspondiente, OTR, ver Seccion 3.

5. VENTAJAS DE ADA EN PROBLEMAS DE TIEMPO REAL

El ambiente natural de ADA es el Tiempo Real y de alli que sea en la especificacion de este tipo de problemas donde

se logra la mayor ventaja frente a los demas metodos de especificacion. Una nocion basica del Tiempo Real tal como una interrupcion presenta un desafio casi imposible para todos los metodos tradicionales de especificacion. Consideremos tres descripciones de una interrupcion: la descripcion verbal, el diagrama y la especificacion en ADA. El resultado sera claro. Comencemos por la descripcion verbal empleada en 1978:

"Las microcomputadoras empleadas en la Central CTA poseen una unica interrupcion implementada y corresponde a la realizada por el reloj de tiempo real cada 1 milisegundo. De acuerdo con esto, la programacion puede ser clasificada en tres areas:

1. PROGRAMAS DE INICIALIZACION: ocurren al comienzo y son desencadenados por una señal de reset (de hardware) o por el encendido de las fuentes de la microcomputadora. En el caso del periferico existe, ademas, un reset por programa.

2. PROGRAMAS DE INTERRUPCION: atienden la interrupcion, cuando esta habilitada, segun los contadores internos del caso. En el caso de la central, las alternativas principales ocurren segun el contador N de interrupciones o segun la señal BATRA del reloj. En el periferico, las alternativas estan reguladas por tres contadores de interrupciones de caracteristicas diferentes: CONTA, CONT5 y RGPAR.

3. PROGRAMAS PRINCIPALES: son interrumpidos por el reloj de tiempo real y atienden una tarea que puede ser postergada."

Una descripcion equivalente se encuentra en el diagrama que se adjunta, tomada de [1].

Finalmente, consideremos la especificacion escrita en ADA:

```
package PROCESADOR_CENTRAL is

with DATOS;              -- estructura de datos

with    CONSTANTES;      -- estructura de constantes

with    IO;              -- estructura de entrada/salida

     task BACKGROUND;

     task FOREGROUND is

          entry SINCRONISMO;
```

```
            entry MILISEGUNDO;

        end FOREGROUND;

    end PROCESADOR_CENTRAL ;
```

Esta especificacion no solamente nos remite a los paquetes de CONSTANTES y de DATOS de la Seccion 3 (el paquete IO no se cita en el presente trabajo) sino que establece claramente el numero de tareas y de puntos de encuentro entre ellas. Todo esto sin necesidad de entrar en detalles de implementacion.

ADA permite mucho mas. Consideremos el siguiente cuerpo para el paquete recien definido:

```
package body PROCESADOR_CENTRAL is

    procedure INIC ;      -- inicializacion

    procedure PCOM ;      -- loop principal

    procedure INTE ;      -- nucleo de tiempo real

    task body BACKGROUND is
    begin
        INIC;                          -- inicializar
        FOREGROUND.SINCRONISMO;        -- sincronizar
        PCOM;                          -- operar
    end BACKGROUND;

    task body FOREGROUND is
        -- declaracion de la interrupcion
        for MILISEGUNDO use at 16#38#;
    begin
        -- espera por sincronismo con el background
        accept SINCRONISMO;
        -- operacion de la interrupcion
```

```
            loop
                accept MILISEGUNDO;
                INTE;
            end loop;
        end FOREGROUND;

    -- el cuerpo del paquete es vacio
    -- se emplea para desencadenar las tareas
    begin
        null;
    end PROCESADOR_CENTRAL;
```

En este cuerpo se identifican claramente los diferentes segmentos de programacion, sus nombres y su interaccion. El punto de "rendezvous" SINCRONISMO permite activar el sistema de interrupciones. La interrupcion MILISEGUNDO -cuya rutina de atencion se encuentra en la direccion 38 hexadecimal- desencadena el procedimiento INTE. Como podemos apreciar, sobre este paquete que contiene unas pocas lineas, se puede continuar la espcificacion, modulo a modulo, de arriba hacia abajo, de una manera natural, mucho mas precisa que cualquiera de los metodos citados anteriormente y con prescindencia de los detalles de implementacion.

El problema considerado, sin embargo, era muy simple. Una unica interrupcion y un esquema de sincronizacion elemental. En el ejemplo que sigue se pone de manifiesto toda la potencia formal de ADA para manejar problemas de tiempo real. La siguiente es una descripcion de las rutinas de atencion de las lineas de la Central Telex:

```
    package PAL is
        task type E ;       -- atencion de lineas
        RUTINA_DE_LINEA:  array(0..N-1) of E ;
    end PAL;
    package body PAL is
        null;       -- entran en actividad las tareas
```

```
end PAL;
```

En este fragmento, de una brevedad insuperable, se declara que existen N replicas -una por linea- de una tarea generica E y que todas estas tareas entran en actividad simultaneamente. Por supuesto, se omite la especificacion de E que no es nada simple, pero este fragmento revela toda la estructura que es necesario implementar, independiente de los detalles superfluos.

6. VENTAJAS DEL EMPAQUETAMIENTO

La posibilidad de estructurar la especificacion en paquetes se traduce, en los hechos, en la correspondiente estructuracion de la implementacion. Por esta razon el diseñador debe usar fuertemente esta posibilidad.

Los usos de los paquetes van mas lejos, sin embargo. En los casos que se emplee mas de un lenguaje de programacion, una division adecuada de paquetes permite realizar una division adecuada de lenguajes de programacion. Mas aun, si se implementa la frontera entre ellos, la comunicacion es posible en forma directa.

Un caso muy importante es el empleo de **simuladores** que permiten ensayar la programacion sin tener completo el sistema. En estos casos basta con invocar un paquete desde un programa que realiza la comunicacion con el operador. De esta manera se puede simular una parte compleja de la programacion. Asi por ejemplo, la secuencia de estados de linea --que posee varios cientos de estados- puede ser simulada mediante un programa ADA como el que sigue:

```
procedure SIMULADOR_DE_LINEAS is

-- declaracion de constantes y variables

N          : constant integer :=4  ;  -- solo 4 lineas

ADE, ADS   : array(0..N-1) of BAUDOT;

LDE        : array(0..N-1) of ESTADO_DE_LINEA;

           := ( 0..N-1 => ELB );

function GET_BAUDOT return BAUDOT is separate;

begin

-- mensaje inicial
```

```
loop

     -- ingreso de la nueva entrada

     for l in 0..N-1 loop

          ADE(l) := GET_BAUDOT;

     end loop;

     -- generar una nueva salida

     ATENCION_DE_LINEAS (ADE(l), ADS(l), LDE(l));

     -- escribirla

end loop;

end SIMULADOR_DE_LINEAS;
```

Este programa invoca el procedimiento **ATENCION_DE_LINEAS** que posee, a su vez, una especificacion ADA presentada en la Seccion 4. Nada impide que, una vez realizada la implementacion definitiva en el lenguaje elegido, se construya la comunicacion con el lenguaje ADA y se proceda a trabajar con el simulador, ahora en la implementacion final. De esta manera se pueden detectar errores por comparacion entre la especificacion y la implementacion final. Esta es una herramienta de mucha utilidad en el desarrollo de la programacion.

7. IMPLEMENTACION DE LA ESPECIFICACION

Con la tecnica propuesta la implementacion se convierte en una forma de "compilacion" de la especificacion, pero realizada en forma no algoritmica. Los implementadores actuan como "compiladores inteligentes" y obtienen asi un producto muy superior al que obtendria un compilador. Como es natural, el esfuerzo humano invertido se traduce en un codigo final mas eficiente en tiempo, en memoria o en el empleo de cualquiera de los recursos criticos que se desee optimizar.

A la inversa, en algunos casos fue necesario ir desde "abajo hacia arriba" y traducir la implementacion en un programa ADA para poder verificar su correccion. Los sistemas de despacho de tareas y otros aspectos de mutua exclusion en un nucleo de Tiempo Real requieren demostrar que son correctos, no alcanza con el simple ensayo.

La experiencia nos muestra que la tesis de demostrar la validez de los programas se convierte en una verdadera necesidad en ciertos aspectos de la programacion de tiempo

real donde intervienen problemas de concurrencia. Tambien aqui ADA es una ayuda importante para abstraer los aspectos esenciales del problema.

8. BALANCE DE LA EXPERIENCIA REALIZADA

La experiencia realizada parte de la idea de la necesidad de una especificacion formal para los problemas de informatica tal como Hoare ha expresado tan bien:

"This activity will culminate in a complete, unambiguous, and provable consistent specification for the entire product." [11]

El resultado muestra que, ademas de esta finalidad teorica y posiblemente todavia lejana, aparecen resultados practicos en mas de un aspecto.

La especificacion realizada en ADA mejora claramente la programacion final a pesar de saber que no se emplearon todas las posibilidades del lenguaje. En el caso considerado, los lenguajes de implementacion fueron C y Assembler y el resultado obtenido tiene **un estilo** de programacion claramente superior a los resultados obtenidos por metodos no formales de especificacion. Sin duda aqui hay una apreciacion que es parcialmente subjetiva pero compartida por todos los miembros del equipo de trabajo.

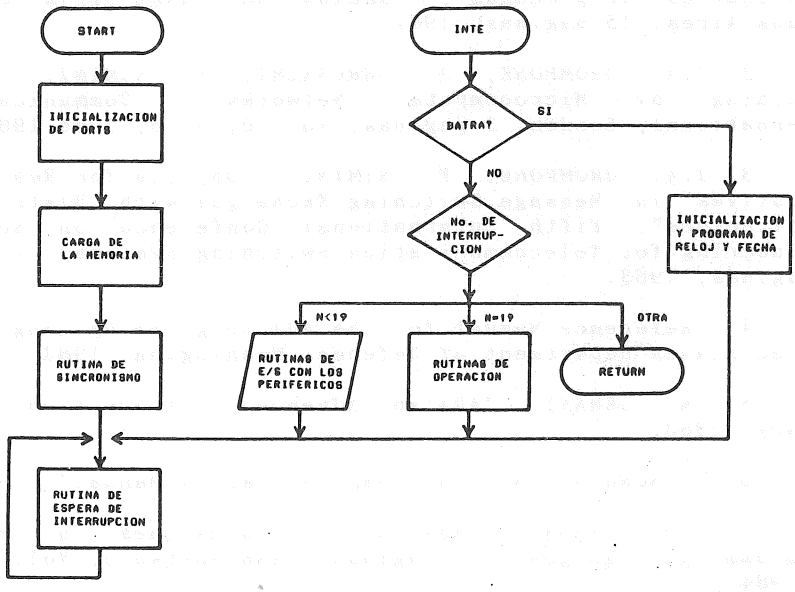
En segundo lugar, una especificacion formal **es una manera de comunicar** entre si los miembros de un equipo de implementacion de un proyecto, sobre todo si este equipo es grande y de orientacion diferente. En nuestro caso se debian conciliar aspectos de Hardware y de Software con personas de formacion en ambas areas. ADA fue un buen punto de encuentro porque si bien es un lenguaje de alto nivel permite expresar claramente, y en el mismo lenguaje, ambos extremos.

En tercer lugar, la experiencia de especificacion en ADA permitio una activa deteccion de errores. Por un lado, se detectaron **errores viejos** que llevaban varios años y que no habian sido advertidos. Por otro lado, se tiene la conviccion que **todos los errores importantes fueron detectados al redactar el documento de especificacion.** La prueba de este hecho es que grandes sub-sistemas resultaron sin errores. Los errores encontrados en la implementacion provenian, casi siempre, de errores en descripciones verbales de documentos relacionados con el proyecto.

Sin duda se dara un enorme paso adelante en la Ingenieria del Software cuando los documentos que **debieran ser formales** realmente lo sean: manuales de componentes, manuales del sistema operativo, manuales de los compiladores

de los lenguajes, etc. quienes fueron la principal fuente de errores de la experiencia.

Estos resultados de utilidad inmediata justifican que en proximos proyectos se continue con la experiencia de desarrollar especificaciones formales escritas en ADA. Es opinion del autor que este es el camino para adquirir la capacidad necesaria en este complejo aspecto de la ingenieria del Software. Cuando la actividad de especificar formalmente sea una practica corriente y bien conocida, recien se podra avanzar hacia la meta propuesta de **demostrar** la correccion de las especificaciones y de las implementaciones de la programacion. A pesar de que los resultados obtenidos son muy alentadores, tambien resulta claro que existe un largo camino de desarrollo tecnologico por delante.



Programas del Procesador Central

BIBLIOGRAFIA

[1] J.A. GROMPONE, N.M.MACE. "Una Central Telex con Procesadores Triplicados", Anales de PANEL'81/12 JAIIO, Buenos Aires, 15 paginas, 1981.

[2] J.A. GROMPONE, J. JERUSALMI, F. SIMINI. "Telex Switching by Microcomputer Networks", Communications International, London, 3 paginas, Vol 10, N. 6, June 1983.

[3] J.A. GROMPONE, F. SIMINI. "Objects for Real Time Executives in Message Switching Exchanges with Distributed Architecture", Fifth International Conference on Sofware Engineering for Telecommunication Switching Systems, London, 5 paginas, 1983.

[4] "Reference Manual for the ADA Programming Language", United States Department of Defense, Washington, 1983.

[5] N. GEHANI. "ADA an Advanced Introduction", New Jersey, 1983.

[6] D. NORRIS. "A (Ada) Compiler User's Manual", 1984.

[7] M. SHAW. "Abstraction Techniques in Modern Programming Languages", 16 paginas, IEEE Software, Vol. 1 No. 4, 1984.

[8] C.A.R. HOARE. "The Emperor's Old Clothes", 1980 ACM Turing Award Lecture, Communications of the ACM, 9 paginas, Vol. 24, No. 2, 1981.

[9] B.A. WICHMANN. "Is ADA Too Big? A Designer Answers the Critics", Communications of the ACM, 6 paginas, Vol. 27 No. 2, 1984.

[10] J. GROMPONE, F. SIMINI. "An Analysis of Errors Detected During the Design and Implementation of a Real Time Multiprocessor system", enviado para publicar en el IEEE Transactions on Computers, 1985.

[11] C.A.R. HOARE. "Programming: Sorcery or Science?", 14 paginas, IEEE Software, Vol. 1, No. 2, 1984.

METODOS DE COMPRESION BIGRAMATICA POSICIONAL

Fco. Javier Gurruchaga Vázquez
Colaboradores: *Félix Ares de Blas, Julio González Abascal, J. Luis García de Madinabeitia*
Facultad de Informática de San Sebastián
San Sebastián - *España*

INTRODUCCION

En nuestros días, se podría decir que la informática forma parte de nuestras vidas. Casi el cien por cien de la información que manejamos y que día a día pasa por nuestras manos, ha sufrido algún proceso de tipo informático.

Imprentas, agencias de información, en general cualquier tipo de empresas, están dando un profundo cambio en sus métodos, estructuras y formas de trabajo, debido a la introducción de los ordenadores, ya que con ellos tenemos información rápida, de fácil manejo y con un menor costo.

Esto nos está llevando, por otro lado, a un problema de tipo informático como es el del almacenamiento de tanta información dentro del ordenador, que se refleja sobre todo en tres puntos a tener en cuenta:

- Costo de almacenamiento.
- Manejabilidad de los datos.
- Búsqueda de los datos.

Además de todo esto, debemos de tener en cuenta que los ficheros o bases de datos, debido a su diseño inicial y a las posteriores ampliaciones, llega un momento en el que dicho fichero tendrá información repetida y espacios vacíos (sin información).

Ante todos estos problemas que hemos de tener en cuenta a la hora de poner en marcha una infraestructura de información, debemos intentar optimizar el tamaño de los diferentes ficheros mediante una codificación que nos lo permita.

A esta reducción mediante una codificación apropiada, se le da el nombre de COMPRESION. La compresión de ficheros nos proporciona una serie de ventajas tales como:

- Disminución de los volúmenes de información.
- Menor cantidad de dispositivos necesarios para el almacenamiento de datos.
- Mayor rapidez en la búsqueda y el proceso de los datos.
- Mayor rapidez de transferencia entre periféricos y memoria, así como, entre terminal y ordenador, lo que nos da mayor economía.

Asímismo, la compresión de ficheros tiene una serie de inconvenientes que requerirán un estudio a la hora de su implantación:

- Obtención, la primera vez, de los diccionarios de compresión.
- Proceso de compresión-descompresión.

Por último, este estudio sólo trata de mejorar un método de compresión que es el bigramático, haciéndolo aún más rentable a la hora de almacenar información en un ordenador.

DEFINICIONES USADAS EN ESTE TRABAJO

TEXTO FUENTE: Texto inicial del que se parte.

CARACTER: Es cada elemento de un alfabeto fuente.

ALFABETO FUENTE (A_f): Conjunto de caracteres con los que podemos formar un texto fuente. $A_f = a_1, a_2, \ldots\ldots, a_n$

TEXTO CODIFICADO: Conjunto de símbolos resultantes de aplicar el método de compresión a un texto fuente.

SIMBOLO o PALABRA CODIGO: Es cada elemento del alfabeto codificador.

ALFABETO CODIFICADOR (A_c): Conjunto de símbolos con los que se puede formar un texto codificado. En este alfabeto se incluyen los caracteres del alfabeto fuente más los símbolos que sustituyen a los caracteres fuente.

$$A_c = b_1, b_2, \ldots\ldots, b_n, \ldots\ldots, b_m$$

PALABRA FUENTE: Serie de caracteres del alfabeto fuente limitados a ambos lados por blancos.

$$\text{Palabra fuente} = \not\beta a_1 \_ a_5 \_ a_7 \not\beta$$

PALABRA CODIFICADA: Serie de símbolos limitados a ambos lados por blancos, de la forma:

- Un primer símbolo del alfabeto codificador que sustituye a caracteres iniciales de una palabra del texto fuente.
- Puede existir un segundo símbolo del alfabeto codificador que sustituye a caracteres finales de una palabra del texto fuente.
- Pueden existir uno o más símbolos del alfabeto codificador que sustituyen a caracteres intermedios de una palabra del texto fuente.

$$\text{Palabra codificada} = \not\beta b_{2i} \not\beta$$
$$\text{Palabra codificada} = \not\beta b_{2i} \_ b_{3f} \not\beta$$
$$\text{Palabra codificada} = \not\beta b_{2i} \_ b_{3m} \_ b_{3f} \not\beta$$
$$\text{Palabra codificada} = \not\beta b_{2i} \_ b_{3m} \_ \ldots\ldots \_ b_{2m} \_ b_{3f} \not\beta$$

BLOQUE MULTILETRA FINAL (BMF): Conjunto de caracteres fuente que se sustituyen por un solo símbolo del alfa-

beto codificador.

FRECUENCIA ABSOLUTA DE BMF (FABMF): Número de veces que un BMF aparece en el texto fuente.

FRECUENCIA RELATIVA DE BMF (FRBMF):

$$FRBMF = \frac{FABMF}{\Sigma\ FABMF + RESTO}$$

RESTO: Caracteres de las palabras fuente que no han sido sustituídos.

COMPRESION ABSOLUTA (CA): Es la frecuencia absoluta de un BMF multiplicada por el número de caracteres menos uno del BMF.

CA = FABMF x (número caracteres BMF - 1)

COMPRESION RELATIVA (CR):

$$CR = \frac{\text{núm. bits texto fuente} - \text{núm. bits texto codificado}}{\text{núm. bits texto fuente}}$$

NUMERO DE BITS POR LETRA A LA SALIDA:

$$\text{núm. bits/letra salida} = \frac{\text{núm. bits texto codificado}}{\text{núm. caracteres texto fuente}}$$

RENDIMIENTO DE UN BMF (RBMF): Es la frecuencia absoluta del BMF multiplicada por el número de caracteres del BMF menos uno.

RBMF = FABMF x (número caracteres BMF - 1)

METODO DE COMPRESION BIGRAMATICA POSICIONAL

Podemos definir la bigramática posicional como una técnica para obtener cadenas de bigramas en un texto atendiendo a la posición que ocupan estos bigramas dentro de una palabra.

Diremos que la compresión bigramática posicional toma como base el método de compresión bigramática tradicional para crear otro sistema muy similar que además de tener en cuenta todas las características de la bigramática tradicional, tiene en cuenta la posición de los caracteres dentro de las palabras de un texto escrito en un idioma natural.

Podríamos preguntarnos, porqué añadir el concepto de posicionalidad. La respuesta se desprende de la tesis doctoral

de Félix Ares [ARES-83] en la que se demuestra que la entropía de un carácter varía según la posición que éste ocupe dentro de la palabra, por lo que podemos utilizar estas diferencias a la hora de eliminar redundancias.

Todas la experiencias que hemos realizado con diversos textos han sido siempre sin intentar introducir el carácter blanco, separador de palabras, dentro de ninguno de los bloques de bigramas -BMFs-. El motivo de tomar esta decisión ha sido el poder comparar nuestros resultados con trabajos previos [GARCIA-80].

Al tener en cuenta la posicionalidad se observa que las palabras se pueden considerar compuestas de tres bloques de símbolos: iniciales, finales e intermedios. Por ejemplo, FERNANDEZ se podría descomponer como:

- Bloque inicial ..... : FERN
- Bloque final ..... : EZ
- Primer bloque intermedio ..... : AN
- Segundo bloque intermedio ..... : D

Por lo tanto, una palabra código dependiendo de la posición que ocupe dentro de la palabra fuente tendrá distintos significados.

COMPARACION DE RESULTADOS ENTRE LA BIGRAMATICA Y LA BIGRAMATICA POSICIONAL.

Para comparar los resultados se ha tomado, como se ha dicho anteriormente, el texto utilizado por E. García Camarero y L. Bengoechea Martínez en su estudio "Un método para la compresión de textos" [GARCIA-80]. Ello ha sido debido a que éste es uno de los pocos trabajos publicados sobre el castellano y la codificación bigramática.

El texto fuente está formado por una secuencia de apellidos castellanos que tiene 14.266 caracteres y su alfabeto fuente ha sido: (A B C D E F G H I J K L M N Ñ O P Q R S T U V W X Y Z - Ø). Como alfabeto codificador se tomaron 125 símbolos. El número de blancos no reducibles dentro del texto es de 1.921.

Teniendo en cuenta lo expuesto anteriormente y utilizando un algoritmo de compresión bigramática, la longitud final del texto codificado fue de 8.356 símbolos. Con lo que se consiguió una reducción del 41,43% medida en caracteres.

En nuestro estudio se utiliza el mismo texto fuente aunque por errores mecanográficos de transcripción del texto, éste ha quedado con 14.240 caracteres. El alfabeto fuente es el mismo. Como alfabeto codificador se tienen las tres tablas:

- Una con BMFs iniciales
- Otra con BMFs finales, y
- Otra con BMFs intermedios.

Tomamos 128 símbolos para cada una de las tablas anteriores.

Para codificar los 28 caracteres del alfabeto fuente nos bastarían 5 bits, pero tomamos la decisión de ampliarlo a 6 bits, para dar la oportunidad de utilizar un alfabeto fuente más amplio. Asimismo, se va a utilizar un código de 7 bits para cada símbolo del alfabeto codificador, ya que, cada tabla va a contener 128 símbolos, de los que los 29 primeros serán los correspondientes a los caracteres del alfabeto fuente.

Con todo lo expuesto se presentan los resultados en la Figura I.

Fijémonos en la columna 11 de dicha figura. Vemos que nuestro método da una mejora del 6% sobre el método bigramático puro. Si nos fijamos en la columna 6 que nos da la compresión relativa en bits, vemos que la diferencia aumenta siendo del 7%. Esta diferencia todavía sería mayor si tuviéramos en cuenta que a todos los símbolos del texto codificado les hemos dado 7 bits, cuando realmente los símbolos denominados como RESTO podrían haberse codificado con sólo 6 bits.

También tenemos que tener en cuenta que se ha partido de un texto fuente con 1.921 blancos no reducibles, con lo que el número de caracteres reducibles en el texto fuente sería de 12.319 (14.240 - 1.921).

Si ahora efectuamos los cálculos sólo sobre los caracteres reducibles, se obtiene:

Aplicando la fórmula de compresión en caracteres:

$$CR = \frac{(14.240 - 1.921) - (7.486 - 1.921)}{(14.240 - 1.921)} = 0,5482$$

Según la bigramática posicional

$$CR = \frac{(14.266 - 1.921) - (8.356 - 1.921)}{(14.266 - 1.921)} = 0,4787$$

Según la bigramática pura

Aplicando la fórmula de compresión en bits

$$CR = \frac{(14.240 - 1.921) \times 6 - (7.486 - 1.921) \times 7}{(14.240 - 1.921) \times 6} = 0,4729$$

Según la bigramática posicional

$$CR = \frac{(14.266 - 1.921) \times 6 - (8.356 - 1.921) \times 7}{(14.266 - 1.921) \times 6} = 0,3918$$

Según la bigramática pura

La diferencia entre ambos métodos, en caso de aplicar la fórmula de compresión en caracteres, es que el método bigra-

mático posicional da una mejora del 6,95%, es decir, un 0,95% más que la calculada en la columna 11, mientras que al aplicar la fórmula de compresión en la que operamos a nivel de bits, la mejora es del 8,11%, es decir, un 1,11% mejor que los resultados calculados en la columna 6.

COMPRESION BIGRAMATICA POSICIONAL POR RENDIMIENTO

A la vista de los resultados que se obtenían en el método anterior, se intuía una mejora del algoritmo, si, en vez de seleccionar los BMFs por frecuencia, se seleccionaban por rendimiento.

El rendimiento tiene en cuenta la frecuencia y la longitud, variables ambas que forman parte del cálculo de la compresión.

La base del algoritmo es el método de compresión bigramática, teniendo en cuenta la posicionalidad de los caracteres dentro de las palabras. Si a todo esto, añadimos que la selección de los bloques que se comprimen se hace por máximo rendimiento, se puede conseguir un algoritmo que obtenga unos bloques con mejor rendimiento que el método de compresión bigramática posicional, ya que así la compresión del texto será mayor.

Las palabras, al igual que en el método de compresión bigramática posicional, se podrán componer de tres bloques de símbolos: iniciales, finales e intermedios. Cada uno de estos bloques es el resultado de aplicar el algoritmo de compresión bigramática posicional por rendimiento al texto fuente en tres fases:

- Empezando por la izquierda de la palabra.
- Empezando por la derecha de la palabra.
- Aplicándolo a los restantes caracteres de la palabra.

El algoritmo es el mismo para las tres fases, dado que antes de comenzar la segunda fase, se ponen los caracteres que no han sido comprimidos en la primera fase al revés, es decir, el último carácter de la palabra se pondrá el primero, el penúltimo carácter se pondrá el segundo, y así sucesivamente, hasta terminar con todos los caracteres de la palabra. Para pasar a la tercera fase haremos lo mismo con los caracteres que no han sido comprimidos en la segunda fase.

DESCRIPCION DEL ALGORITMO DE COMPRESION DE BIGRAMATICA POSICIONAL POR RENDIMIENTO.

1.- ELEMENTOS DEL ALGORITMO.

Para realizar el algoritmo vamos a necesitar tres tablas.

- TABLA F: Esta tabla es una matriz de dos dimensiones en la que cada fila esta formada por una palabra del texto fuente, junto con su frecuencia de aparición dentro de ese texto. La matriz tendra tantas filas como palabras

distintas tenga el texto y estará ordenada alfabéticamente.

- TABLA R: Esta tabla es también una matriz de dos dimensiones en la que cada fila esta formada por los distintos rendimientos que se puedan obtener de todos los BMFs posibles de cada palabra. También cada fila contendrá además una columna con el mayor valor de todos ellos y otra columna tendrá su posición dentro de esta fila.

- TABLA S: Esta tabla contendrá por cada BMF que el algoritmo seleccione el símbolo a sustituir y el rendimiento que se obtiene. Los N primeros símbolos serán los correspondientes a los caracteres del alfabeto fuente.

2.- FORMACION DE LA MATRIZ F.

La matriz F va a estar formada por todas las palabras diferentes que componen el texto.

Cada carácter va a ser un elemento $a_{i,j}$, donde i = 1,2,.....,n y j = 1,2,.....,m. El índice i nos indica la posición de la palabra de que se trata y el índice j la posición de cada carácter dentro de la palabra.

La matriz tendrá otras dos columnas:

- Columna de frecuencias: $f_i = a_{i,m+1}$, donde cada f_i será la frecuencia de aparición de la palabra i.
- Columna de máxima longitud: $l_i = a_{i,m+2}$, donde cada l_i indicará la longitud (en número de caracteres) de la palabra i.

Ejemplo Figura II.

3.- FORMACION DE LA MATRIZ R.

Esta matriz va a estar formada por todos los rendimientos $r_{i,j}$ que se obtienen de todos los BMFs posibles de cada palabra.

El cálculo de los $r_{i,j}$ se hará:

$r_{i,j} = P * (j - 1)$ siendo

$P = \sum_{x=s}^{t} f_x$, siempre que se cumpla que:

$a_{s,1}-a_{s,2}-\cdots-a_{s,j}=a_{s+1,1}-a_{s+1,2}-\cdots-a_{s+1,j}=\cdots=$

$= a_{t,1}-a_{t,2}-\cdots-a_{t,j}$

Además, los $r_{i,j}$ que cumplan las siguientes condiciones tendrán el valor cero:

- los $r_{1,j}$
- los $r_{i,j}$ pertenecientes a la misma fila que cumplan que:

  para j = k,k+1,.....,m-1

  donde m = l_i y k sería el menor número que cumpla una de las dos ecuaciones siguientes:

$a_{i-1,1}-a_{i-1,2}-\cdots-a_{i-1,k-1} = a_{i,1}-a_{i,2}-\cdots-a_{i,k-1}$

$a_{i,21}-a_{i,2}-\cdots-a_{i,k-1} = a_{i+1,1}-a_{i+1,2}-\cdots-a_{i+1,k-1}$

Esto es, si dentro de una palabra hay distintos BMFs de igual frecuencia sólo le calcularemos el rendimiento al BMF de más caracteres, el resto tendrá valor cero.

- los $r_{i,j}$, siendo j = k,k+1,.....,l y siendo l = q-1, donde q es el número más bajo tal que $a_{i,q} \neq a_{i+1,q}$, y siendo k el número más pequeño que cumple que $a_{i-1,k} \neq a_{i,k}$, siendo $k \neq q$. Ello es debido a la misma razón que el punto anterior, pero referido a un BMF perteneciente a varias palabras.

Esta matriz tendrá también dos columnas más:

- Columna de máximo rendimiento:

  $rm_i = r_{i,m+1}$, donde cada rm_i será el máximo $r_{i,j}$ de los calculados en la fila i.

- Indice de columna de máximo rendimiento:

  $xm_i = r_{i,m+2}$, donde cada xm_i contendrá la posición j del máximo $r_{i,j}$ de la fila i.

Ejemplo Figura IV

Una vez formadas las tablas F y R, se pasará a desarrollar el algoritmo para calcular los BMFs de mayor rendimiento, para lo cual se irán variando los valores de la tabla R en función de los BMFs que se vayan eligiendo.

4.- DESARROLLO DEL ALGORITMO.

A.- Escoger de entre todos los valores de rm el máximo. A igualdad entre dos filas escoger la fila con l mayor. A igualdad de l, escoger el primero de entre éstos últimos.

B.- Incorporaremos a la tabla S el BMF escogido con su correspondiente rendimiento. Cada BMF de esta tabla se corresponderá con un símbolo del alfabeto codificador. Ejemplo Figura III.

C.- Recalculamos los $r_{i,j}$ en los cuales el BMF escogido esté totalmente embebido.

Lo haremos de la siguiente forma:

1.- Sea $r_{u,v}$ el rendimiento máximo.

Haremos $r_{i,j} = 0$, $\forall$ i que cumpla:

$$a_{i,1}-a_{i,2}-\cdots-,a_{i,v} = a_{u,1}-a_{u,2}-\cdots-a_{u,v}$$

2.- Recalculamos los $r_{i,j}$ que sean:

$\forall$ j > v y $\forall$ i que cumpla:

$$a_{i,1}-a_{i,2}-\cdots-,a_{i,v} = a_{u,1}-a_{u,2}-\cdots-a_{u,v}$$

mediante la fórmula:

$r_{i,j} = P \times (j - v)$ donde

$P = \sum_{x=s}^{t} f_x$, siempre que se cumpla:

$$a_{s,1}-a_{s,2}-\cdots-a_{s,j}=a_{s+1,1}-a_{s+1,2}-\cdots-a_{s+1,j}=$$

$$=\cdots=a_{t,1}-a_{t,2}-\cdots-a_{t,j}$$

En el apartado 1 ponemos a cero todos los $r_{i,j}$ que ya no van a formar parte de la selección, dado que se ha

encontrado un $r_{i,j}$ mejor que ellos y que los aglutina. En el apartado 2 recalculamos los $r_{i,j}$ del resto de la palabra que no quedan cubiertos por el BMF escogido. El rendimiento se obtiene sustituyendo el BMF escogido por un símbolo, con lo cual la longitud varía, lo que hace que haya un cambio de rendimiento que se deberá volver a calcular.

Ejemplo: si FERNAN en la palabra FERNANDA lo sustituímos por "*" nos queda *DA con lo que el $r_{i,j}$ de *DA es:

8 x (3 - 1) = 16 (Frecuencia x (longitud - 1))

Con lo cual la nueva tabla R nos quedaría como en la Figura V.

D.- Recalculamos los $r_{i,j}$ en los cuales el BMF escogido está parcialmente embebido, es decir, los $r_{i,j}$ cuyo BMF asociado sea subconjunto del BMF escogido.

Estos subconjuntos deberán ser de la siguiente forma:

1.- El BMF subconjunto tendrá más de dos caracteres.

2.- Se deberá cumplir que:

$$a_{i,1}-a_{i,2}-\cdots-a_{i,j} = a_{e,1}-a_{e,2}-\cdots-a_{e,j}$$

siendo i el índice del subconjunto y e el índice del BMF escogido.

Ejemplo: al escoger el BMF FERNAN, los BMFs a los que éste puede afectar en sus $r_{i,j}$ serán: FE, FER, FERN, FERNA.

El recálculo de los rendimientos de estos BMFs se hará quitando la parte en que el BMF escogido afectaba a éstos. (Figura VI).

E.- Seguidamente volveremos al punto A de este algoritmo y así estaremos en un ciclo hasta que ya no queramos escoger más BMFs.

Las Figuras IV a XI muestran como se iría desarrollando el algoritmo. En la Figura III, sumando la columna de rendimientos, obtenemos la compresión absoluta del texto (CA) si utilizamos un símbolo para sustituir cada uno de

los BMFs de la tabla.

COMPARACION DE RESULTADOS.

En este apartado vamos a comparar los resultados obtenidos al utilizar el algoritmo de compresión bigramática posicional por rendimiento de dos formas:

- Con los resultados que obtuvimos con el algoritmo de compresión bigramática posicional y

- Con los resultados obtenidos por el método de compresión bigramática tradicional.

En ambos casos los resultados se compararán siempre con el fichero de apellidos que venimos usando habitualmente.

En las pruebas realizadas con el algoritmo de compresión bigramática posicional por rendimiento el texto fuente utilizado es el mismo con 6 caracteres más, que forman una fecha que se le introdujo al texto, por lo que, con los errores de transcripción mas estos 6 caracteres, el texto fuente ha quedado constituido por 14.246 caracteres.

Como alfabeto codificador se tienen tres tablas:

- Una con BMFs iniciales
- Otra con BMFs finales y
- Otra con BMFs intermedios.

Cada una de estas tablas tiene, para poder comparar con los anteriores estudios, 128 símbolos codificados cada uno con 7 bits, de los que los 29 primeros serán los correspondientes a los caracteres del alfabeto fuente.

Los caracteres del alfabeto fuente se codificarán con 6 bits cada uno.

Para comparar los resultados obtenidos nos fijaremos en la Figura XII en la que están expuestos todos ellos.

Si observamos las columnas 6 y 7 de dicha figura, que nos dan la compresión relativa, una calculada en base a bits y la otra en base a caracteres, vemos la mejora que nos da este algoritmo frente a los anteriores.

En la columna 6 se ve una mejora del 7,42% frente al método de compresión bigramática posicional, y del 14,42% frente al método de compresión bigramática pura.

En la columna 11 esta mejora se cifra en un 6,35% frente al método de compresión bigramática posicional, y en un 12,35 frente al método de compresión bigramática pura.

Asimismo el número de bits que harían falta para codificar un carácter del texto fuente baja a 3,23 bits en este método, frente a los 3,68 bits que nos harían falta aplicando el método de compresión bigramática posicional y

frente a los 4,10 bits necesarios en el método de compresión bigramática pura.

Todo ésto se ha hecho teniendo en cuenta que los blancos no son reducibles, ya que si éstos formaran parte de los BMFs y por lo tanto fueran reducibles, entonces el índice de compresión aumentaría de la siguiente forma:

A.- Aplicando la fórmula de compresión relativa en bits:

$$CR = \frac{14.246 \times 6 - (6.584 - 1.921) \times 7}{14.246 \times 6} = 0,6181$$

B.- Aplicando la fórmula de compresión relativa en caracteres:

$$CR = \frac{14.246 - (6.584 - 1.921)}{14.246} = 0,6726$$

En la Figura XIII se han extraido los resultados obtenidos cuando el número de símbolos es igual en las tres tablas. En estos tres casos hubiera sido muy fácil el insertar los blancos dentro de los BMFs tanto para su selección como para su codificación-decodificación. Queremos resaltar que hemos obtenido a pesar de ello, una compresión del 63% y que el número de bits necesarios para codificar un carácter ha sido, como ya hemos dicho anteriormente, de 2,96. Este resultado sería bastante superior si los blancos hubieran entrado a formar parte de los BMFs.

CONCLUSIONES

En este apartado de conclusiones, vamos a tratar de resumir las aportaciones que este estudio puede dar al campo de la compresión de textos o ficheros de apellidos escritos en castellano.

1.- Se ha desarrollado a partir del método de compresión bigramática un nuevo método al que se le ha agregado la idea de seleccionar los bigramas atendiendo a la posición que éstos ocupan dentro de las palabras.

2.- Se ha comprobado que la selección de bloques multiletra nos da un mejor rendimiento si la selección se hace primero escogiendo los BMFs iniciales y luego los finales, que si lo hacemos al revés, es decir, escogiendo primero los BMFs finales y luego los iniciales.

3.- Se ha demostrado que la compresión bigramática posicional da un mayor índice de compresión que la bigramática clásica cuando usamos un número de palabras código aproximadamente igual en los diccionarios de transcodificación.

4.- Se ha comprobado que la compresión bigramática posicional con posterior selección por rendimiento, da en la mayoría de los casos, mejor índice de compre-

sión que si aplicamos tan sólo el método de compresión bigramática posicional sin ningún tipo de selección.

5.- Se ha comprobado que este tipo de compresión no da tan buen rendimiento en ficheros de tipo literario donde los monosílabos constituyen casi un 30% del texto. Este rendimiento, aumenta considerablemente en textos donde este tipo de palabras prácticamente no aparecen como bien pueden ser, los ficheros de apellidos.

6.- Ante la necesidad de introducir una nueva variable - longitud - al método de compresión bigramática posicional, se ha desarrollado un nuevo algoritmo, al que se le ha dado el nombre de algoritmo o método de compresión bigramática posicional por rendimiento.

7.- El algoritmo de compresión bigramática posicional por rendimiento trata de buscar bloques multiletra lo más óptimos posibles, haciendo un análisis parcial de la palabra: es decir, sólo se tiene en cuenta o bien la parte inicial de la palabra para seleccionar bloques iniciales, o bien la parte final de la palabra para seleccionar bloques finales, o la parte intermedia de la palabra para seleccionar bloques intermedios.

8.- Se ha comprobado como este último método supera a los métodos de compresión bigramática tradicional y al de compresión bigramática posicional, en índice de compresión, bien se mida en caracteres o en bits. También los supera en el número de bits que son necesarios para codificar un carácter del texto fuente.

9.- Como conclusión y resumen final, se puede decir que se ha desarrollado un método de compresión, partiendo de la bigramática y de la posición de los caracteres dentro de la palabra, al que se le ha llamado bigramática posicional, obteniendo mejores resultados en ficheros de apellidos castellanos, que la bigramática clásica. Al final, se ha unido a esto la idea original de tener en cuenta el rendimiento a la hora de contruir los diccionarios. Y se le ha llamado bigramática posicional por rendimiento y los resultados son aún mejores que los de los dos métodos anteriores.

BIBLIOGRAFIA

[ARES-83] Ares de Blas, Félix, "Un método para disminuir la redundancia en la transmisión de textos escritos en castellano". Tesis doctoral. F.I.S.S. 1.983.

[GARCIA-80] García Camarero, Ernesto y Bengoechea Martínez, L. "un método para la compresión de textos". Boletín del

Centro de Cálculo de la Universidad Complutense. Número 36. Junio 1.980.

[GURRUCHAGA-85] Gurruchaga, J.,G. de Madinabeitia, J., Gonzalez Abascal, J. y Ares de Blas, F., "Un método de compresión bigramática posicional". Revista de Informática y Automática. Num. 63. PP. 32-35. Enero-Marzo 1.985.

[GURRUCHAGA-86] Gurruchaga, J.,G. de Madinabeitia, J., Gonzalez Abascal, J. y Ares de Blas, F., "Un método de compresión bigramática posicional por rendimiento". Revista de Informática y Automática. Aceptado para su publicación.

[GURRUCHAGA-86] Gurruchaga, J. "Nuevos métodos de compresión bigramática". Tesis de Licenciatura. FISS 1.986.

[GURRUCHAGA-85] Gurruchaga, J.,G. de Madinabeitia, J., Gonzalez Abascal, J. y Ares de Blas, F., "Entropy of Word Position". Revista IEEE ejemplar Mayo-1.985.

[HAMILTON-80] Hamilton, D.A., Herrold P.R., Ossefort M.J., "Digramatic text Compression". IBM Technical Disclosure Bulletin. Vol. 23. N. 2- Julio 1.980.

[HUFFMAN-52] Huffman, D.A., "A method for the construction of Minimun Redundancy Codes". Proc. IRE. 40. PP. 1.098-1.101. 1.952.

[RODRIGUEZ-78] Rodríguez, Prieto, Amador, "Algoritmo de compresión de datos. Una aplicación práctica". Inforprim 78. Madrid, Nov. 1.978.

[WELCH-84] Welch, Terri A., "A technique for High Performance Data Compression". IEEE-Computer. Vol 17, N. 6. June 1.984.

	1	2	3	4	5	6	7	8	9	10	11
	NUM. BMFs INIC.	NUM. BMFs FINAL	NUM. BMFs INTER.	NUM. BITS TEXTO FUENTE	NUM. BITS TEXTO CODIF.	COMPR. RELAT. (en bits)	NUM. BITS/ LETRA SALIDA	COMPR. ABSOL. BMFs INIC.	COMPR. ABSOL. BMFs FINAL.	COMPR. ABSOL. BMFs INTER.	COMPR. RELAT. (en carac)
Bigramatica Posicional	128	128	128	85.440	52.402	0,3866	3,6799	5.063	1.102	589	0,4743
Segun E.G.C y L.B.M	128	---	---	85.596	58.492	0,3166	4,1001	5.910	---	---	0,4143

FIGURA I

i \ j	1	2	3	4	5	6	7	8	9	10	f	l
1	F	E	R	M	I	N					5	6
2	F	E	R	N	A	N					10	6
3	F	E	R	N	A	N	D	A			8	8
4	F	E	R	N	A	N	D	E	Z		8	9
5	F	E	R	N	A	N	D	I	T	O	5	10
6	F	E	R	N	A	N	D	O			6	8
7	G	A	R	C	I	A					5	6
8	G	A	R	M	E	N	D	I	A		10	9
9	G	O	I	C	O	E	C	H	E	A	15	10
10	G	O	N	Z	A	L	E	Z			5	8
11	G	O	N	Z	A	L	O				5	7

FIGURA II

Simbolo	BMF	Rendim.
α	FERNAN	185
β	GOICOECHEA	135
γ	GARMENDIA	80
δ	GONZAL	50

FIGURA III

Para i=1,2,...,11 y

j=1,2,...,10

i \ j	1	2	3	4	5	6	7	8	9	10	rm	xm
1		5	10			25					25	6
2											0	0
3							27	16			27	7
4							27		24		27	7
5							27			20	27	7
6							27	12			27	7
7		15	30			25					30	3
8		15	30						80		80	9
9											0	0
10		10				50		35			50	6
11		10				50	30				50	6

FIGURA VIII

i \ j	1	2	3	4	5	6	7	8	9	10	rm	xm
1		5	10			25					25	6
2											0	0
3							27	16			27	7
4							27		24		27	7
5							27			20	27	7
6							27	12			27	7
7		15	30			25					30	3
8											80	9
9											0	0
10		10				50		35			50	6
11		10				50	30				50	6

FIGURA IX

i \ j	1	2	3	4	5	6	7	8	9	10	rm	xm
1		5	10			25					25	6
2											0	0
3							27	16			27	7
4							27		24		27	7
5							27			20	27	7
6							27	12			27	7
7		5	10			25					25	6
8											0	0
9											0	0
10		10				50		35			50	6
11		10				50	30				50	6

FIGURA X

i \ j	1	2	3	4	5	6	7	8	9	10	rm	xm
1		5	10			25					25	6
2											0	0
3							27	16			27	7
4							27		24		27	7
5							27			20	27	7
6							27	12			27	7
7		5	10			25					25	6
8											0	0
9											0	0
10								10			50	6
11							5				50	6

FIGURA XI

i \ j	1	2	3	4	5	6	7	8	9	10	rm	xm
1		42	84			25					84	3
2		42	84			185					185	6
3		42	84			185	162	56			185	6
4		42	84			185	162		64		185	6
5		42	84			185	162			45	185	6
6		42	84			185	162	42			185	6
7		15	30			25					30	3
8		15	30						80		80	9
9		25								135	135	10
10		25				50		35			50	6
11		25				50	30				50	6

FIGURA IV

i \ j	1	2	3	4	5	6	7	8	9	10	rm	xm
1		42	84			25					84	3
2											185	6
3							27	16			185	6
4							27		24		185	6
5							27			20	185	6
6							27	12			185	6
7		15	30			25					30	3
8		15	30						80		80	9
9		25								135	135	10
10		25				50		35			50	6
11		25				50	30				50	6

FIGURA V

i \ j	1	2	3	4	5	6	7	8	9	10	rm	xm
1		5	10			25					25	6
2											0	0
3							27	16			27	7
4							27		24		27	7
5							27			20	27	7
6							27	12			27	7
7		15	30			25					30	3
8		15	30						80		80	9
9		25								135	135	10
10		25				50		35			50	6
11		25				50	30				50	6

FIGURA VI

i \ j	1	2	3	4	5	6	7	8	9	10	rm	xm
1		5	10			25					25	6
2											0	0
3							27	16			27	7
4							27		24		27	7
5							27			20	27	7
6							27	12			27	7
7		15	30			25					30	3
8		15	30						80		80	9
9										135	135	10
10		25				50		35			50	6
11		25				50	30				50	6

FIGURA VII

	1	2	3	4	5	6	7	8	9	10	11
	NUM. BMFs INIC.	NUM. BMFs FINAL	NUM. BMFs INTER.	NUM. BITS TEXTO FUENTE	NUM. BITS TEXTO CODIF.	COMPR. RELAT. (en bits)	NUM. BITS/ LETRA SALIDA	COMPR. ABSOL. BMFs INIC.	COMPR. ABSOL. BMFs FINAL.	COMPR. ABSOL. BMFs INTER.	COMPR. RELAT. (en carac)
Bigramatica Posicional Rendimiento	128	128	128	85.476	46.087	0,4608	3,2350	5.448	1.558	656	0,5378
Bigramatica Posicional	128	128	128	85.440	52.402	0,3866	3,6799	5.063	1.102	589	0,4743
Segun E.G.C y L.B.M	128	---	---	85.596	58.492	0,3166	4,1001	5.910	---	---	0,4143

FIGURA XII

1	2	3	4	5	6	7	8	9	10	11
NUM. BMFs INIC.	NUM. BMFs FINAL	NUM. BMFs INTER.	NUM. BITS TEXTO FUENTE	NUM. BITS TEXTO CODIF.	COMPR. RELAT. (en bits)	NUM. BITS/ LETRA SALIDA	COMPR. ABSOL. BMFs INIC.	COMPR. ABSOL. BMFs FINAL.	COMPR. ABSOL. BMFs INTER.	COMPR. RELAT. (en carac)
256	256	256	85.476	42.264	0,5055	2,9667	6.939	1.627	397	0,6291
128	128	128	85.476	46.087	0,4608	3,2350	5.448	1.558	656	0,5378
64	64	64	85.476	50.142	0,4133	3,5197	3.957	1.320	612	0,4133

FIGURA XIII

CRITERIOS PARA EL ESTUDIO DE FACTIBILIDAD DE UN SISTEMA DE TRADUCCION COMPUTARIZADA

Iván Guzmán de Rojas
La Paz - *Bolivia*

1. Explicación de conceptos

En el presente documento entenderemos por "traducción" el proceso de conversión de un texto escrito en un lenguaje de entrada L_0 a varios textos en lenguajes de salida L_1, L_2, L_3, etc.; de tal manera que los textos de salida reflejen segmento por segmento el mismo mensaje contenido en el texto de entrada.

Aquí entendemos por "segmento" una secuencia de caracteres que parten, ya sea desde el comienzo de un texto, o siguiendo a un signo de puntuación, hasta llegar a otro signo de puntuación. En esta definición no interesa si esa secuencia representa palabras con algún significado. Lo que intentamos con ella es definir términos aptos para diseñar programas capaces de traducir textos por computadora.

Como "signos de puntuación" consideramos solamente los siguientes: "." ";" "?" "¡". Las comas, comillas y paréntesis se consideran como caracteres especiales dentro de un segmento.

Los segmentos así definidos pueden constituír oraciones en el sentido gramatical, títulos o subtítulos que aparecen en el texto; también pueden estar conformados por una secuencia de símbolos no traducibles (gráficas, caracteres de control de edición, etc.).

Durante el proceso de traducción, segmento por segmento, se va traduciendo el texto de entrada y generando los correspondientes textos de salida. No siempre un segmento, consistente en una oración del lenguaje de entrada se convierte a un segmento en todos los lenguajes de salida; hay oraciones de entrada que se requiere desglosar en dos o tres oraciones, por tanto segmentos, en alguno de los lenguajes de salida.

Una capacidad que se espera de todo sistema traductor, es la de tratar segmentos intraducibles como los que se presentan en textos que vienen grabados como documento para procesador de textos; estos segmentos son simplemente transferidos a los textos de salida (sin conversión) para preservar códigos que controlan la estructura de edición del documento.

Entenderemos por "léxico" la base de datos en la que se encuentran almacenados los términos de un lenguaje correctamente enlazados con los correspondientes términos de significado equivalente en los otros lenguajes; es decir, hablamos de una base de datos multilingüe. Toda consulta a ella debe ser posible por cualquiera de los lenguajes que la constituyen, obteniendo inmediatamente las equivalencias en los demás lenguajes.

Al establecer las relaciones de equivalencia en el léxico, frecuentemente se presenta el fenómeno de homonimia lexical, consistente en la existencia de más de un significado para un término en un determinado lenguaje. Por ejemplo, el término "vino" en español, podría significar en inglés, la forma verbal "came" o también el sustantivo "wine". Todo sistema de traducción computarizada debe poder resolver estas homonimias de manera que en el texto traducido aparezca el término equivalente correcto como para representar fielmente el mensaje contenido en el segmento de entrada.

Según las formas de resolver el problema de homonimia, ésta se clasifica en diferentes tipos de homonimia, correspondientes a la técnica utilizada en los algoritmos de discriminación.

En los casos de "homonimia semántica", el algoritmo discriminador requiere necesariamente de información semántica. Por ejemplo la oración en inglés "She plays tennis" exige que se discrimine el verbo "to play", ya sea como "tocar" (un instrumento musical) o como "jugar" (un juego). El discriminador puede actuar siempre y cuando sea capaz de "entender" la oración en por lo menos algo de su significado; es decir, su léxico debe contener información semántica que relacione "to play" en el sentido de "tocar" con los sustantivos que designan un instrumento musical,

asimismo, el término en el sentido de "jugar" con las designaciones de juegos, como tennis.

En cambio, en otras situaciones no es necesario que el discriminador entienda la oración para resolver una homonimia. Por ejemplo, en la "homonimia sintáctica", un analizador sintáctico con la capacidad de darse cuenta cuál es la secuencia apropiada que deben seguir los términos en una oración, podrá fácilmente resolver casos como: "Esa mujer que vino ayer quería comprar vino". Este ejemplo es de solución inmediata, si el conocimiento gramatical del discriminador es suficiente como para saber que después del subordinador "que" no puede seguir un sustantivo, así como después del verbo conjugado "quiere beber" no puede seguir un verbo; el término adecuado en ambas posiciones se determina por exclusión.

Dado el alto valor asignado a los sistemas de traducción computarizada, desafortunadamente no se cuenta con información detallada sobre los algoritmos con que trabajan los discriminadores utilizados en los sistemas existentes. Esta situación conspira contra toda posibilidad de realizar comparaciones y constataciones de los adelantos científicos logrados en este campo.

Entonces, al hablar de "traducción computarizada", no nos referimos simplemente a búsquedas electrónicas en un léxico en medio magnético, sino que nos referimos a un proceso de conversión de segmentos capaz de lidiar, por lo menos parcialmente, con los casos de homonimia lexical.

Otro aspecto implícito en el término "traducción" es obviamente la capacidad del traductor de manejar la transformación sintáctica que conlleva el reordenamiento de palabras acorde con las reglas gramaticales en los lenguajes de salida. Un sistema de conversión de segmentos sin reordenamiento sintáctico se reduce a una consulta de diccionario, quizás con la sola ventaja del análisis morfológico previo. Está comprobado que los sistemas que solamente traducen textualmente resultan ser muy poco prácticos, ya que los costos de reordenamiento manual para obtener la traducción final superan los de la traducción por el método convencional, a pesar de la ventaja de búsqueda rápida en diccionario.

Además del correcto reordenamiento sintáctico y de la discriminación de homonimias, está sobreentendido que los algoritmos de búsqueda en todo sistema de traducción computarizada deben estar dotados de eficaces analizadores y sintetizadores morfológicos, capaces de normalizar las palabras que se presentan en el texto original, antes de ejecutar la consulta al léxico, a fin de evitar tener que almacenar todas las variantes morfológicas de una palabra, en especial las flexiones verbales y las declinaciones por caso, género y número de sustantivos y adjetivos.

La capacidad de resolver casos de ambigüedad por medio de un análisis en contexto aumenta aún más la calidad del texto traducido en bruto. Sin embargo, este requerimiento exige de técnicas costosísimas en la programación del sistema. Por ejemplo, para resolver la ambigüedad de la oración en inglés: "This paper is excellent", requiere que el sistema sepa por el contexto en que se encuentra la oración, si se está hablando de "paper" en el sentido de papel o en el sentido de una publicación científica. En este caso la homonimia del término "paper" no puede resolverse tratando el segmento aisladamente del resto del texto. Hasta qué punto vale la pena el esfuerzo de programación para resolver este tipo de problemas de incidencia relativamente baja, es una cuestión muy discutible al tratarse de la traducción asistida por computadora, en que de todos modos se cuenta con la intervención humana. Un sistema de traducción completamente automática, por el momento utópica, tendría que resolver este tipo de umbigüedades también.

Dependiendo de la forma en que el sistema está programado para manejar las gramáticas, podemos distinguir entre dos clases de sistemas radicalmente diferentes entre sí. Por un lado tenemos aquellos en que las gramáticas, en especial las reglas sintácticas de los diversos lenguajes, se encuentran prescritas en las instrucciones mismas del programa. Estos traductores de gramática interna tienen la desventaja de que sus programas componentes requieren ser constantemente modificados a medida que se desea mejorar su capacidad de manejo gramatical.

Por otro lado tenemos sistemas como ATAMIRI, en que las gramáticas son tratadas de modo externo al programa. Las especificaciones gramaticales, inclusive las reglas sintácticas y las reglas de transformación de un lenguaje a otro, se almacenan en tablas de codificación externas (diccionarios gramaticales) que se van enriqueciendo independientemente de la estructura del programa traductor. Los sistemas con diseño de gramática externa pueden llegar a tener la habilidad de servirse de un cierto número de estructuras básicas para deducir a partir de ellas otras que aparecen por primera vez en nuevos textos a traducir.

Estos sistemas que gozan de la capacidad de inferencia y de aprendizaje de reglas gramaticales los llamaremos "traductores inteligentes", ya que el manejo externo de las gramáticas les confiere un cierto nivel de inteligencia artificial.

Los traductores inteligentes ofrecen la ventaja económica de bajos costos en el proceso de implementación del sistema, ya que facilitan enormemente el proceso de enriquecimiento de diccionarios, tanto para optimizar su competencia gramatical como su nivel de "conocimiento" lexical.

2. Las operaciones básicas de traducción.

Al igual que en el proceso de traducción por métodos convencionales, en el caso de un sistema asistido por computadora, siguen siendo necesarias las operaciones básicas.

a) Control lexicográfico
b) Manejo de textos en ambiente multilingüe
c) Traducción en bruto (borrador)
d) Afinado (revisión, corrección y pulido)

Al utilizar la computadora como ayuda para la traducción, todavía se hace necesaria una operación básica adicional:

e) Transcripción del texto original del papel a medio magnético.

Este proceso adicional podría resultar económico si se utiliza un lector óptico, que convierte el texto "leído" del papel a un texto grabado en soporte magnético, en un archivo de procesador de textos.

Si bien es cierto que las operaciones básicas son las mismas en ambos casos, dependiendo de cómo es el diseño y la programación del sistema asistido por computadora, la forma en que estas operaciones se llevan a cabo puede ser muy diferente de la usual.

Para comenzar, el manejo de textos en ambiente multilingüe requiere de las técnicas de procesamiento de textos más avanzadas, a fin de que el trabajo de afinado se desarrolle eficientemente.

Cuando el texto generado por el programa traductor es de alta calidad, para el trabajo de afinado, simplemente se requieren funciones de procesamiento de textos aptas para efectuar modificaciones en los segmentos de salida, comparando la entrada con la salida (de modo bilingüe); eventualmente, con la capacidad de consulta interactiva al léxico, para ayudar a resolver homonimias o seleccionar adecuadamente términos sinónimos.

En cambio, si en la traducción en bruto no se encuentran bien resueltos los reordenamientos sintácticos, será necesario con el apoyo de aquellas funciones de procesamiento de textos aptas para realizar ágilmente movimientos de palabras o grupos de palabras dentro el segmento.

La eficacia con que el sistema computarizado maneja los textos en la tarea de afinación es determinante para la obtención de un buen factor de aumento de productividad. Es más, posiblemente la tarea de traducción con la sola ayuda de un eficaz procesador de textos bilingüe, aún sin contar con el borrador de la traducción por máquina, puede llegar a ser ventajoso para que el traductor profesional genere su propio borrador.

Las tareas de control lexicográfico adquieren características muy diferentes de las usuales con los métodos convencionales. Con un traductor computarizado el terminologista tiene a su disposición una base de datos multilingüe, en la que puede efectuar consultas, no solamente en orden alfabético, sino también por grupos semánticos, ricamente clasificados en su tesaurus electrónico. La posibilidad de ejercer un eficaz control del inventario de términos asegura la consistencia lexical de las traducciones, convirtiendo el trabajo lexicográfico en una actividad de nuevas dimensiones, incomparable con los métodos corrientes.

Por ejemplo, el módulo lexicográfico del sistema ATAMIRI es capaz de llevar el inventario de términos facilitando las entradas nuevas en modo interactivo. El programa revisa previamente la base de datos lexical para verificar posibles repeticiones, garantizando así un alto grado de consistencia lexical, sobre todo en el dominio de la terminología técnica especializada.

Todo buen sistema lexicográfico debe permitir el control de entradas en modo interactivo, reduciendo al mínimo la utilización de costosos y morosos procesos de trabajo sobre incómodos listados en papel, con las búsquedas de términos faltantes a simple vista.

Obviamente la ventaja más importante de un sistema de traducción computarizado, consiste en la generación electrónica del primer borrador (traducción en bruto). Un ordenador es capaz de llevar a cabo esta tarea a una alta velocidad, que puede ir de las 2.000 a las 50.000 palabras por hora, según el sistema que se utilice.

Una alta velocidad de generación tiene la ventaja de permitir realizar varios tratamientos de un mismo texto, antes de proceder con la afinación, facilitando las tareas de enriquecimiento lexical de modo previo al tratamiento final, gracias al analizador de términos faltantes (en el primer tratamiento).

Cuando el sistema traductor genera un texto en bruto de alta calidad, requiriendo pocas correcciones fáciles de efectuar, ofrece una doble ventaja:

a) La traducción en bruto ya puede servir como material de trabajo, en situaciones de urgencia, en que interesa más la entrega inmediata que la calidad del borrador.

b) La entrega rápida del borrador permite disponer de más tiempo para el trabajo de afinado y análisis lexical.

En un buen sistema de traducción asistida por computadora, necesariamente se consiguen bajos costos en las operaciones básicas tomadas globalmente, en comparación con los métodos convencionales; además, está la ventaja de la entrega más rápida. Por ejemplo, en una prueba piloto utilizando el sistema ATAMIRI, en aproximadamente 1.000 páginas de texto (manuales técnicos) se logró obtener una traducción en bruto de tal calidad, que la tarea de afinado se efectuó tan rápidamente, que el mismo traductor profesional produjo ocho veces el volumen que conseguía procesar con el método convencional, en la misma unidad de tiempo. La generación de la traducción en bruto tomó un tiempo insignificante (6 horas) a una velocidad de 30.000 palabras por hora.

La forma en que se llevan a cabo las operaciones en un sistema computarizado, tiene un impacto determinante en el modo de organizar y administrar la operación de traducción de documentos, así como en la manera de seleccionar y adiestrar personal. Estos aspectos no se discuten en el presente documento porque no inciden directamente en el cálculo de costos. Sin embargo deben tomarse en cuenta en todo estudio completo para encarar un proyecto de traducción asistida por computadora.

Como se verá en los próximos capítulos, es crucial para determinar la factibilidad de un proyecto de traducción computarizada, calcular el valor del factor de aumento de productividad que conlleva el nuevo sistema. Este factor depende fundamentalmente de la eficacia con que trabajan los programas del sistema, resolviendo los problemas esbozados anteriormente.

Por esta razón, todo estudio de factibilidad deberá necesariamente evaluar las técnicas empleadas por el sistema en consideración, verificando que las operaciones básicas se ejecutan de modo completo y dando solución a los problemas que presenta el lenguaje natural. En lo posible debe constituír parte de la evaluación un período de prueba del sistema, previo taller de trabajo que permita al personal de traductores familiarizarse con el sistema y efectuar mediciones de competencia gramatical y rendimiento en el trabajo de afinado de la traducción en bruto. Cualquier consideración de tipo económico, que previamente no esté sustentada por la solidez del sistema a emplearse, corre el riesgo de quedar como una simple especulación de lo que podría lograrse.

Para una operación de prueba eficaz, es suficiente escoger textos dentro de un determinado ámbito de trabajo, selectivamente almacenar el léxico para ese tipo de documentos (del orden de 10.000 palabras), y producir unas 200 páginas de traducción. El tiempo requerido para una operación piloto preliminar es del orden de tres a cuatro meses. Una rápida evaluación en base a simples demostraciones de un paquete de programas puede resultar muy riesgosa.

3. La economía de la traducción computarizada

Aquí entendemos por "traducción computarizada" el proceso antes descrito, en el que un conjunto de programas de computadora asisten al traductor profesional en las diversas tareas requeridas por el trabajo completo de traducción masiva de textos de un lenguaje de entrada a varios lenguajes de salida.

Un modelo económico adecuado para este proceso necesariamente debe tener en cuenta los siguientes parámetros:

V Volumen de páginas de texto traducido que se supone el sistema será capaz de traducir en un período de T años (tiempo de explotación del sistema).

V/T Volumen anual de producción (págs/año).

I Inversión de capital total requerida para poner en marcha el sistema, que se espera crecerá al cabo de T años al monto:

$$I_T = I(1+r)^T$$

donde r es la tasa de retorno al capital que se espera obtener del proyecto. El crecimiento del capital resulta del negocio de venta de traducciones, de modo que también es:

$$I_T = PU - G \quad \text{donde}$$

P es el precio unitario ($/pág) de la traducción producida por el sistema.

G es el gasto global incurrido en el período T para mantener en operación el sistema y cubrir los costos de comercialización.

De las dos anteriores fórmulas obtenemos que:

$$I = (PU - G)/(1+r)^T$$

Si no se quiere desperdiciar recursos en un proyecto a pérdida, la inversión máxima permitida sería la que corresponde a una tasa de retorno nula (r=0), es decir:

$$I_{max} = PU - G$$

Si bien es cierto que el mercado de traducciones en el mundo es enorme, no deja de ser finito, asimismo, el precio P está fijado por el mercado y los gastos G dependen de los costos de operación tanto para generar la traducción en bruto como para su afinado ("post-editing"). Por lo tanto, la inversión máxima admisible para un proyecto de traducción computarizada no es ilimitadamente grande.

Para fines de comparación con un proyecto basado en un sistema de traducción convencional, para atender el mismo mercado al precio P, en un volumen V, por un período de explotación T, definimos el factor de aumento de productividad F, por la relación:

$$F = G_C/G$$

donde G_C son los gastos de operación y comercialización cuando se trabaja con el método convencional (no computarizado). El factor F necesariamente debe ser mayor que 1 si suponemos que el sistema computarizado eleva la productividad del traductor profesional y facilita las comunicaciones para alcanzar el mercado. De esta manera obtenemos, referida a los gastos convencionales, que la inversión máxima admisible es:

$$I_{max} = PV - G_C/F$$

Igualmente, si se espera una tasa de retorno a la inversión de capital en investigación y desarrollo de sistemas, la inversión que hace "factible" el proyecto no puede sobrepasar el monto:

$$I = (PV - G_C/F)/(1+r)^T$$

Estas fórmulas sintetizan el análisis "macroeconómico" de la traducción computarizada, vista como un proceso industrial. Si se desea, los parámetros G y G_C podrían desmenuzarse en una serie de componentes en detalle; sin embargo, estas expresiones nos permiten apreciar nítidamente cuáles son los factores que juegan el rol decisivo en la determinación de la factibilidad de un proyecto de traducción computarizada.

El factor de aumento de productividad F puede ser medido efectivamente en una operación de prueba de cualquier sistema que se proponga. Básicamente se trata de comparar la productividad de un traductor profesional utilizando el sistema con su productividad "normal" (con el método convencional). En los sistemas de traducción asistida por computadora, en realidad F no puede exceder el valor de 20, que corresponde al caso en que la tarea de "post-editing" se reduce simplemente a la lectura de revisión de la traducción en bruto, sin actuar corrigiéndolo; la velocidad de lectura del ser humano es el límite. En los sistemas existentes, se han observado valores para F que van de 2 a 14, dependiendo de la calidad del sistema utilizado y del tipo de texto con que se efectúa la prueba.

Para obtener cifras estimativas, resulta práctico expresar las anteriores fórmulas en términos de costos unitarios, sean:

$C_I = I/V$ el costo en \$/pág de la reposición del capital de inversión;

$C_G = G/V$ el costo en \$/pág de cobertura de gastos de operación y comercialización utilizando el sistema computarizado;

$C_{GC} = G_C/V$ el costo en \$/pág de la cobertura de gastos de operación y comercialización trabajando con los métodos convencionales (respecto a los que se desea comparar el sistema computarizado).

Así, dividiendo la última fórmula por el volumen V, deducimos:

$$C_I = (P - C_{GC}/F)/(1+r)^T$$

Puesto que esta fórmula se refiere a valores unitarios, también nos puede servir para evaluar la adquisición de programas para utilizarlo en traducción computarizada. Efectivamente, podemos considerar C_I como el costo de reposición de la inversión en ese paquete de programas, más el costo de arranque del sistema (antes de entrar en productividad).

Al aplicar estas fórmulas en un estudio de factibilidad no debe dejar de considerarse que el valor de la tasa de retorno a la inversión r, nunca puede ser menor que una tasa de intereses bancarios, de lo contrario se entra en el terreno del despilfarro de recursos; pues, es preferible tener el dinero en el banco, ganar intereses y con eso dar trabajo a traductores profesionales en vez de malgastarlo en computadoras inservibles. Por otro lado, en los cálculos no es aconsejable suponer períodos de explotación del paquete de programas demasiado largos; el mercado ofrece cada vez nuevas opciones, cada vez más versátiles; por ello, un período T=5 años es lo más realista.

Al realizar el estudio de factibilidad, necesariamente se debe evaluar el programa que se tiene previsto utilizar, exigiendo pruebas de rendimiento que permitan medir efectivamente el factor de aumento de productividad F, con un texto típico en el área de aplicación del usuario, de por lo menos unas 200 páginas.

El precio P debe considerarse de modo realista, de acuerdo a la oferta de servicios de traducción especializada; el mercado internacional actual, se mueve por los 20 a 45 $/pág, según el caso.

Para el estudio de factibilidad en centros de traducción de empresas o instituciones internacionales que cuentan con su propio equipo de traductores, no se debe utilizar para P el valor de precio de mercado, puesto que el precio "real" con el que operan es mucho mayor, dada las ventajas que trae la asignación de prioridades, privacidad y otras, al disponer de su propio servicio de traducción. Usualmente el "precio interno" en estos centros de traducción es del orden de 90 a 120 $/pág. dependiendo de los niveles de sueldo de los traductores profesionales y de las facilidades de equipos e instalación de oficina de que hacen uso.

En realidad para estos casos, P es el valor que tiene C_{GC} según la contabilidad de la empresa o institución para la que se efectúa el estudio de factibilidad; sin embargo, es un valor difícil de determinar cuando no se cuenta con una estadística de producción anual por traductor. De nada sirve trabajar con el rendimiento esperado teóricamente de un traductor, en base a páginas traducidas por hora, puesto que todo profesional tiene muchas horas en que no traduce, sino más bien investiga terminología, asiste a cursos de entrenamiento técnico, se enferma, toma vacaciones, etc. Como una cifra estimativa se puede considerar que la producción anual de un traductor profesional de tiempo completo es del orden de mil páginas.

4. Ejemplos ilustrativos y discusión de resultados

Para ilustrar la aplicación de los criterios que sugerimos para un estudio de factibilidad de un sistema de traducción computarizada, a continuación discutimos dos ejemplos de cálculo estimativo.

En el primer caso, sea la cuestión el determinar la inversión máxima admisible para un mercado de un millón de páginas que se quiere captar en cinco años. Por ejemplo, queremos competir con el proyecto EUROTRA desarrollando un sistema multilingüe para entrar al mercado de la Comunidad Europea. Por tratarse de un proyecto riesgoso esperamos una tasa de retorno del 24.6% anual, es decir, deseamos triplicar el capital en los cinco años.

Considerando un costo anual de 100.000 $ por traductor profesional, que cubre su sueldo y gastos asociados a su trabajo, incluyendo los gastos de comercialización, podemos estimar que el costo unitario C_{GC} = 100 $/pág. Con estas premisas, para tres diferentes valores del factor de incremento de productividad calculamos el costo de reposición de la inversión de capital:

F	C_I ($/pág)
4	-0.33
5	1.33
6	2.47
10	4.67
20	6.33

Para un valor de V = 1 Millón de páginas (en 5 años), la inversión justificable para triplicarse en ese período, es igual a las anteriores cifras de C_I, solamente que leídas en millones de dólares. Por ejemplo, para F = 6, se justifica invertir 2.47 millones de dólares. En otras palabras, el valor de la tecnología de traducción computarizada está perfectamente determinado por el factor F con que permite trabajar. Para los valores supuestos en nuestras premisas, un valor de F = 4, es decir una tecnología que solo permite cuadruplicar la productividad del traductor profesional, traería pérdidas.

Como dato ilustrativo, indicamos que el proyecto EUROTRA, desde su arranque, hasta la conclusión del sistema multilingüe para atender los nueve lenguajes oficiales de la CEE, cuenta con una inversión del orden de 34 millones de dólares. Esta suma puede justificarse si se espera captar en cinco años un mercado del orden de 10 millones de páginas, o si la inversión se considera de baja rentabilidad.

El riesgo en la inversión de capital consiste en que si los resultados de la investigación y desarrollo de "software" y léxico computarizado no conducen a un valor de F adecuado, la tasa de retorno disminuye y se hace difícil captar el mercado. Por ello la teoría de representación del lenguaje natural que se aplique en el proyecto es de importancia decisiva.

Otro caso ilustrativo, de enfoque algo diferente en la premisa del precio, constituye el de una empresa con su propio centro de traducciones, que sin aumentar su personal de traducción desea elevar la producción de modo considerable; por ejemplo, cuadruplicarla. Esta es la situación típica en que una empresa se ve enfrentada a una inusitada avalancha de textos a traducir en corto plazo. La cuestión es determinar hasta qué nivel de gasto anual se puede permitir para cubrir costos de programas y computadora, logrando así aumentar la productividad sin tener que contratar nuevo personal adicional.

Suponiendo que su precio interno de traducción es de 80 $/pág, que la tasa de retorno a la inversión es solamente de 10% anual, para un período de explotación de 5 años, el cálculo resulta ser: $C_I = (80 - 80/4)/(1.1)^5 = 37.27$ $/pág. Es decir, este es el costo unitario que la empresa puede permitirse erogar en programas y máquina para servirse de un sistema de traducción computarizada. Si su volumen de traducciones al que desea llegar anualmente, es del orden de 10.000 páginas año, su presupuesto para cubrir el costo del servicio (programas y computadora, sin incluir otros gastos que tenía anteriormente) puede alcanzar el monto de 370,270 $/año.

Cualquiera que sea el caso cuya factibilidad se desea determinar, resulta fácil un análisis preliminar aplicando las fórmulas anteriormente deducidas, sin tener que entrar en un análisis de costos detallado. Obviamente, si se desea afinar cifras, los parámetros en las fórmulas deberán obtenerse previo estudio de costos desglosados en los diversos rubros que intervienen en toda la actividad de traducción de textos de gran volumen.

CALCULO EFICIENTE DE CONJUNTOS DE LOOKAHEADS LALR(1) EN MICROCOMPUTADORES

Arturo Montes Sinning
Rodrigo López Beltrán
Universidad de los Andes
Bogotá - *Colombia*

1. Introducción

Durante el primer semestre de 1983 se comenzó un proyecto de software, en el departamento de sistemas de la Universidad de los Andes, cuyo objetivo era producir un generador de analizadores sintácticos LALR para ser utilizado como herramienta didáctica en el curso de construcción de compiladores de nuestro currículum. Dadas las restricciones de uso del equipo del Centro de Cálculo de la universidad (en donde ya se disponía de un generador de ese estilo), era necesario implementar el sistema sobre los microcomputadores que soportan buena parte de la atención a estudiantes. Una primera versión [MON85], que se ha venido utilizando desde comienzos de 1985, funciona sobre microcomputadores con sistema operacional MS-DOS.

Las ideas básicas para el desarrollo del proyecto fueron tomadas del trabajo de DeRemer y Pennello[DeR82], pero el resultado final contiene algunas mejoras con respecto al cálculo de los LookAheads (Algoritmo 3.6), aparte de ciertas ideas acerca de información útil para la resolución de los eventuales conflictos que se presentan en una gramática (Sección 4). El documento original de este trabajo [MON85], no resalta claramente las virtudes antes mencionadas pues se pierde en lo intrincado del formalismo utilizado, que es el propuesto en [DeR82]. Este artículo aclara las ideas de [MON85] gracias al excelente formalismo expuesto por Park, Chang y Choe [PAR85]. En las secciones 2 y 3 se exponen los algoritmos de cálculo del autómata LR(0) y los LookAheads, respectivamente, junto con la

introducción del formalismo "a la Park". En la sección 4 se esbozan las ideas con respecto a la resolución de conflictos y en la sección 5 se presentan algunas conclusiones.

2. Generación del Autómata LR(0)

Para la mayoría de las definiciones que aparecen a continuación utilizaremos la terminología habitual (ver [AHO77]). Algunas, que son la base de los resultados presentados en el artículo, han sido tomadas de [PAR85] y [DeR79].

Definición 2.1

Las nociones de símbolo y de cadena se suponen conocidas. Un vocabulario V es un conjunto de símbolos. V^* denota el conjunto de todos las cadenas de símbolos de V. V^+ denota $V^* - \{ lambda \}$, donde lambda es la cadena nula. La longitud de una cadena alfa la denotaremos por |alfa|. El primer símbolo de una cadena alfa lo denotaremos por Prim(alfa); la cadena que sigue al Prim(alfa) será Rest(alfa) (el resto de alfa); el último símbolo lo designaremos por Ult(alfa).

Definición 2.2

Si R es una relación, R^* denota la clausura reflexiva y transitiva de R, y R^+ denota la clausura transitiva de R.

Definición 2.3

Una gramática G, es una cuádrupla G=<N,T,S,P> donde:
- N es un conjunto de símbolos llamados no-terminales.
- T es un conjunto de símbolos llamados símbolos terminales.
- T es disyunto de N, $S \in N$, y $V = N \cup T$.
- P es un subconjunto de N x V^* llamado Producciones de G. Si (A,w) $\in$ P escribiremos A -> w donde A se denomina la parte izquierda y w la parte derecha de una producción.

A continuación establecemos las siguientes convenciones que serán usuales en este artículo a menos que se especifique lo contrario:

S,A,B,C, ...	$\in$	N
X	$\in$	V
t,a,b,c, ...	$\in$	T
..... x,y,z	$\in$	T^*
alfa, beta, gamma ..	$\in$	V^*.

Definición 2.4

$=>_d$ es una relación de V^* en V^* tal que si alfa $\in V^*$ y z $\in T^*$, entonces alfaAz $=>_d$ alfawz si y sólo si A -> w $\in$ P. Al subíndice d lo eliminaremos de la notación, y cuando aparezca el símbolo => entenderemos que se trata de $=>_d$. $=>^*$ y $=>^+$ denotan la clausura reflexiva y transitiva de la relación => y ambas se leerán "deriva".

Definición 2.5

A es un no-terminal anulable si y sólo si, A =>* lambda.

Definición 2.6

alfa es una forma sentencial si y sólo si, alfa ∈ V* y S=>* alfa. Se llamará una sentencia si alfa ∈ T*.

Definición 2.7

L(G), el lenguaje generado por G, es el conjunto de sentencias. En otras palabras, $L(G) = \{ x \in T^* \mid S \Rightarrow^+ x \}$.

Definición 2.8

G es una gramática reducida, si y sólo si, para toda forma sentencial $S \Rightarrow^+$ alfaAbeta, existe y ∈ T* tal que A =>* y.

Definición 2.9

Para una gramática arbitraria G, podemos construir una gramática aumentada G' que contiene la producción S' -> S# en donde S' ∉ N y # es un nuevo símbolo terminal. Ahora L(G')= L(G)# y además se garantiza que el símbolo S' no aparece en la parte derecha de ninguna producción.

En lo sucesivo, supondremos que todas las gramáticas son reducidas y aumentadas.

Definición 2.10

Una tripleta (A,alfa,beta) es un ítem, si y sólo si, existe A -> alfa beta ∈ P, donde (A,alfa,beta) ∈ N x V* x V*. El ítem (A,alfa,beta) será denotado por A -> alfa.beta . Si alfa=lambda lo llamaremos un ítem inicial y si beta=lambda lo llamaremos un ítem final y si alfa y beta son lambda lo denominaremos un ítem nulo.

Definición 2.11

Una Tabla es un conjunto de ítems.

Definición 2.12

Si S es una Tabla,
CLAUSURA(S)= S U { A -> .w| B -> alfa.A beta ∈ CLAUSURA(S)}.

Definición 2.13

Si S es una Tabla y X es elemento de V,

GOTO(X,S) = CLAUSURA({A -> alfa X.beta| A -> alfa.X beta ∈ S}).

Definición 2.14

Si S es una Tabla, KERNEL(S) es el mínimo K, subconjunto de S, tal que CLAUSURA(K) = CLAUSURA(S).

Definición 2.15

Sea S una Tabla, SUCESORES(S) = { GOTO(X,CLAUSURA(S)) ¦ X ∈ V }.

Definición 2.16

TP(G), el conjunto de tablas de parsing LR(0) de una gramática G, es

$$TP(G) = \{ CLAUSURA(\{S \rightarrow .S'\#\}) \} \cup \{ CLAUSURA(K) \mid K \in SUCESORES(K') \text{ y } K' \in TP(G) \}.$$

Definición 2.17

Sea A -> alfa.Xbeta un ítem. La función GEN se define como,

GEN(A -> alfa.Xbeta) = Ø si X ∈ T
ó
A -> alfa.Xbeta es un ítem final.

GEN(A -> alfa.Xbeta) = { X ->.w ¦ X -> w ∈ P } en cualquier otro caso. Por extensión, si S es una Tabla, entonces

GEN(S) = U {GEN*(A -> alfa.Xbeta)¦A -> alfa.Xbeta ∈ S}.

Lema 2.1

Sea A -> alfa.Xbeta un ítem,

CLAUSURA({A -> alfa.X beta}) = GEN*(A -> alfa.Xbeta).

Del lema y de la definición anteriores se observa que el cálculo de la CLAUSURA de una tabla, se puede realizar calculando GEN*. Por otro lado, GEN nos sugiere una relación (que llamaremos I) entre los no-terminales:

B I C si y solamente si B -> Cgamma ∈ P.

Al grafo dirigido asociado con la relación I lo llamaremos el I-grafo.

Lema 2.2

Sea A -> alfa.Bbeta un ítem, entonces

GEN*(A -> alfa.Bbeta) = { C ->.w¦ B I* C }.

Definición 2.18

El Autómata LR(0) para una gramática G (que en adelante

llamaremos ALR(0)) es una cuádrupla ALR(0) = (Q,i°,SUCC,TP(G)) donde

- TP(G) es el conjunto de tablas de parsing de G.
- Q es un conjunto de índices de una enumeración de TP(G).
- i° es el índice del primer elemento de la enumeración de TP(G) y corresponde a CLAUSURA({S' -> .S#}).
- SUCC es una función de Q x V en TP(G) tal que

  SUCC(p,X) = GOTO(X,Cp)

  donde Cp denota el p-ésimo elemento en la enumeración de TP(G). De ahora en adelante cuando referenciemos un elemento Cp de TP(G) nos estaremos refiriendo al p-ésimo elemento de la enumeración de TP(G) y con Kp al KERNEL(Cp). Cuando no se introduzca ambigüedad, utilizaremos un índice p como sinónimo de Cp.

El siguiente algoritmo nos calculará TP(G) para una gramática G cualquiera:

Algoritmo 2.1

```
Begin
  TP(G) := {CLAUSURA({S' -> .S#})}      ;
  For Each K ∈ TP(G) tal que sus SUCESORES no hayan sido
     calculados dO_Loop
     For Each K' ∈ SUCESORES(K) tal que K' ∉ TP(G) dO_Loop
         TP(G) := TP(G) U { K' } ;
     End_Loop ;
     Marcar a K indicando que sus SUCESORES ya han sido
     calculados ;
  End_Loop      ;
End ;
```

Si guardamos en una cola los elementos de TP(G) cuyos sucesores no han sido calculados, obtenemos una primera mejora y el algoritmo quedaría de la siguiente forma:

Algoritmo 2.2

```
QK  :  Es una cola de los KERNELs de los elementos de TP(G)
       cuyos SUCESORES no han sido calculados.

Begin
    QK := <{S' -> .S#}>     ;
    TP(G) := {CLAUSURA({S' -> .S#})}  ;
    WHILE Not(Empty(QK)) dO_Loop
      K := DeQueue(QK)          ;
      For Each K' ∈ SUCESORES(K) tal que K' ∉ TP(G) dO_Loop
          TP(G) := TP(G) U { K' } ;
          EnQueue(K',QK)    ;
      End_Loop  ;
    End_Loop    ;
End     ;
```

Se observa, que la operación de averiguar si una Tabla K se encuentra en TP(G), es determinante en la complejidad del

algoritmo. Por lo tanto, respondiendo a ella de una manera eficiente podemos minimizar el costo del cálculo de la tabla de parsing.

Definición 2.19

Sea A -> alfa.Xbeta un ítem, se define la función ESTADOS como,

ESTADOS(A -> alfa.Xbeta)={ p ∈ Q ¦ A -> alfa.Xbeta ∈ K_p }.

donde K_p es el KERNEL del p-ésimo elemento en la enumeración de TP(G).

Lema 2.3

Sea q ∈ Q, entonces

Kq = { A -> alfa.Xbeta ¦ q ∈ ESTADOS(A -> alfa.Xbeta }.

Definición 2.20

Sea Cp una Tabla de TP(G) y q ∈ Q. Diremos que q es un estado candidato de Cp si y solamente si Kq es un superconjunto de S.

Lema 2.4

q - Q es un estado candidato de S, si y sólo si,

Para todo A -> alfa.Xbeta ∈ S, q ∈ ESTADOS(A ->alfa.Xbeta).

Por el lema anterior tenemos que el costo de responder a la pregunta P ∈ TP(G) es lineal con respecto a #({q¦ q es un estado candidato de P}), que es lo mismo que #({q¦ Kp⊂Kq}).

3. Cálculo de los LookAheads

Definición 3.1

Sea q ∈ Q y alfa ∈ V*, se extiende SUCC a cadenas de V*

SUCC(q,alfa) = q si alfa es lambda,

SUCC(q,alfa) = SUCC(SUCC(q, Prim(alfa)), Rest(alfa)).

Definición 3.2

Sea q ∈ Q, A ∈ N. La pareja (q,A) es una "transición no terminal" si y sólo si SUCC(q,A) está definido.

Definición 3.3.

Sea q ∈ Q y A -> alfa.beta un ítem,

LA(q,A -> alfa.beta) = { t ∈ T¦S' => S# =>* gammaAsigma# =>*
gamma alfa beta sigma# =>*
gamma alfa betatteta =>* x,
x ∈ L(G) y SUCC(i°,gamma,alfa) = Cq }.

Definición 3.4

Sea q ∈ Q y A -> w ∈ P,

LA(q,A -> w) = LA(q,A -> w.).

Definición 3.5

Sea q ∈ Q y t ∈ T,

Reduce(q,t) = { A -> w ∈ P ¦ t ∈ LA(q,A -> w) }.

Definición 3.6

Sea alfa ∈ V*,

FIRST(alfa)= {t ∈ T¦ alfa =>* tw para algún w ∈ T*}.

Si alfa es anulable entonces lambda también pertenece a FIRST(alfa).

Definición 3.7

Sean L, M conjuntos de cadenas,

L @ M = L si lambda no pertenece a L
ó
L @ M = L U M en caso contrario.

Definición 3.8

Sean B y C - N,

PATH(B,C) = U { FIRST(betan...beta2 beta1) ¦ B0=B, Bn=C,
B0 -> B1beta1 ∈ P, B1 -> B2beta2 ∈ P, ...,
B_{n-1} -> $B_n$betan ∈ P }.

Definición 3.9

Sea p ∈ Q y alfa ∈ V*,

PRED(p,alfa) = { q ∈ Q ¦ p = SUCC(q,alfa) }.

Lema 3.1 [PAR85]

Sean p, q ∈ Q tales que q ∈ PRED(p,alfa1), A -> alfa1.alfa2 ∈ Cp, t ∈ LA(p,A -> alfa1.alfa2), A ≠ S. Entonces debe haber por lo menos un ítem B -> beta1.A'beta2 ∈ Kq que satisface la siguiente condición:

t ∈ { a ¦ a ∈ PATH(A',A) @ FIRST(beta2t),
q ∈ PRED(p,alfa1), A' I* A,
B -> beta1.A'beta2 ∈ Kq}.

Teorema 3.1 [PAR85]

Sea p ∈ Q y A -> alfa1.alfa2 ∈ Cp, con A ≠ S, entonces:

LA(p,A -> alfa1.alfa2)={t¦t ∈ PATH(A',A)@FIRST(beta2)@
LA(q,B -> beta1.A'beta2),
q ∈ PRED(p,alfa1), A' I* A, B -> beta1.A'beta2 ∈ Kq}.

Colocando la ecuación anterior en un lenguaje más ameno,

LA(p,A -> alfa1.alfa2)=
U { U { PATH(A',A)@FIRST(beta2)@LA(q,B -> beta1.A'beta2) ¦
A' I* A, B -> beta1.A'beta2 ∈ Kq
} ¦
q ∈ PRED(p,alfa1) }.

Esta ecuación muestra explícitamente la idea principal del algoritmo para el cálculo de los LookAheads.

Algoritmo 3.1

Function LA(p,A -> alfa1.alfa2) Return Set Of T Is

```
Begin
    LA := Ø ;
    For Each q ∈ PRED(p,alfa1) dO_Loop
        For Each B -> beta1.A'beta2 ∈ Kq,A' I* A dO_Loop
            LA := LA U PATH(A',A) ;
            dO_When (lambda ∈ PATH(A',A))
                LA := LA U FIRST(beta2) ;
                dO_When(lambda ∈ FIRST(beta2))
                    LA := LA U LA(q,B -> beta1.A'beta2);
                End_dO ;
            End_dO ;
        End_Loop ;
    End_Loop ;
End ;
```

Se observa, que el algoritmo anterior requiere una modificación para evitar llamadas recursivas al mismo estado con el mismo ítem. Para ello, construimos una relación que nos evite casos como el anterior, de tal manera que dicha relación nos permita, por así decirlo, construir un orden de evaluación para el cálculo de los LAs.

Tomando las ideas propuestas por DeRemer y Pennello [DeR79], redefinimos el conjunto FOLLOW de la siguiente manera:

Definición 3.10

Sea q ∈ Q y A ∈ N,

FOLLOW(q,A)={a|a ∈ FIRST(beta2)@LA(q,B -> beta1.A'beta2),
B -> beta1.A'beta2 ∈ Kq, A I* A'}.

Desprendiéndose el siguiente lema:

Lema 3.2 [PAR85]

Sean p,q - Q y A -> alfa1.alfa2 ∈ Cp,

LA(p,A -> alfa1.alfa2) = { a | a ∈ FOLLOW(q,A),
q ∈ PRED(p,alfa1) }.

Reescribiendo los FOLLOWs con base en los FOLLOWs,

Lema 3.3 [PAR85]

Sea p,q ∈ Q,

FOLLOW(q,A) = { a|a - FIRST(beta2)@FOLLOW(r,B),
B -> beta1.Abeta2 ∈ Kq,
r - PRED(q,beta1),
C -> gamma1.Bgamma2 ∈ Cr }.

Si lambda ∈ FIRST(beta2), tenemos FOLLOW(q,A) incluye al FOLLOW(r,B).

Definición 3.11

(q,A) INCLUDES (r,B) si y solamente si se cumple la condición anterior.

El cálculo de los LAs según este método envuelve evaluar la clausura reflexiva de la relación INCLUDES. El cáculo de los LAs para un ítem final A ->w. de un estado p, es la union de los FOLLOW(q,A) donde q ∈ PRED(p,w). Obsérvese que

FOLLOW(p,A) = U FOLLOW(q,B) tales que (p,A) INCLUDES* (q,B).

Algoritmo 3.2 [DeR82]

```
Construcción del grafo debido a la relación INCLUDES.
For Each q ∈ Q  dO_Loop
        For Each B -> beta1.Abeta2 ∈ Cq dO_Loop
            FOLLOW(q,A) := FIRST(beta2)   ;
            INCLUDES(q,A) := Ø ;
            dO_When(lambda ∈ FIRST(beta2))
                For Each r ∈ PRED(q,beta1)  dO_Loop
                   For Each C -> gamma1.Bgamma2 ∈ Cr dO_Loop
                        INCLUDES(q,A) := INCLUDES(q,A) U { (r,B) }  ;
                   End_Loop    ;
                End_Loop   ;
             End_dO    ;
        End_Loop  ;
    End_Loop   ;
```

Algoritmo 3.3 [DeR82]

Cálculo de los LookAheads usando FOLLOWs,

```
For Each p ∈ Q  dO_Loop
    For Each A -> w. ∈ Kp  dO_Loop
       LA(q,A -> w.) := Ø ;
       For Each  q ∈  PRED(p,w)  dO_Loop
          LA(q,A -> w.) := LA(q,A -> w.) U FOLLOW(q,A) ;
       End_Loop ;
    End_Loop    ;
End_Loop    ;
```

El problema del método de DeRemer y Pennello es que la relación INCLUDES llega a ser demasiado grande, debido a la cantidad de ítems iniciales que pueden hacer aparecer arcos de la misma (en especial cuando hay producciones de la forma A -> B). Podemos obtener una reducción sustancial de dicha relación si sólo adicionamos arcos a través de los ítems presentes en el kernel exclusivamente. Y según el método desarrollado por J.C.H. Park, K. M. Choe y C. H. Chang [PAR85] tenemos,

Lema 3.4 [PAR85]

Sean p,q ∈ Q y A -> alfa1.alfa2 ∈ Cp,

$$LA(p, A \to \alpha_1.\alpha_2) = \{\, a \mid a \in PATH(A', A) @ FOLLOW(q, A'),\ q \in PRED(p, \alpha_1),\ A' \ I^*\ A,\ B \to \beta_1.A'\beta_2 \in Kq \,\}.$$

Lema 3.5 [PAR85]

Sean p,q ∈ Q ,

$$FOLLOW(q, A') = \{\, a \mid a \in FIRST(\beta_2) @ PATH(P', B) @ FOLLOW(r, B'),\ B \to \beta_1.A'\beta_2 \in Kq,\ r - PRED(q, \beta_1),\ B' \ I^*\ B,\ C \to \gamma_1.B'\gamma_2 \in Kr \,\}.$$

Si $\lambda \in FIRST(\beta_2)$ y $\lambda \in PATH(B', B)$, tenemos FOLLOW(q,A') incluye al FOLLOW(r,B').

El algoritmo para generar la relación INCLUDES quedaría:

Algoritmo 3.4 [PAR85]

```
For Each q ∈ Q  dO_Loop
    For Each B -> beta1.Abeta2 ∈ Kq dO_Loop
        FOLLOW(q,A) := FIRST(beta2)    ;
        INCLUDES(q,A) := Ø  ;
        dO_When(lambda ∈ FIRST(beta2))
            For Each r ∈ PRED(q,beta1)  dO_Loop
               For  Each  C -> gamma1.Bgamma2 ∈ Kr,  B' I*  B
               dO_Loop
                  FOLLOW(q,A') := FOLLOW(q,A') U PATH(B',B);
                  dO_When(lambda ∈ PATH(B',B))
                      INCLUDES(q,A) := INCLUDES(q,A) U { (r,B) }
                  End_dO   ;
               End_Loop    ;
            End_Loop   ;
         End_dO    ;
    End_Loop  ;
End_Loop   ;
```

Y el cálculo de los LookAheads,

Algoritmo 3.5 [PAR85]

```
For Each p ∈ Q  dO_Loop
    For Each A -> w. ∈ Kp  dO_Loop
       LA(q,A -> w.) := Ø    ;
       For Each q ∈ PRED(p,w), B -> beta1A beta2 ∈ Kq dO_Loop
           LA(q,A -> w.) := LA(q,A -> w.) U PATH(A',A) ;
           dO_When(lambda ∈ PATH(B',B))
              LA(q,A -> w.) := LA(q,A -> w.) U FOLLOW(q,A) ;
           End_dO   ;
       End_Loop ;
    End_Loop   ;
End_Loop   ;
```

Todos los algoritmos anteriores calculan los FOLLOW de las transiciones no-terminales y construyen completamente la relación INCLUDES. Si se desea implementar dichos algoritmos en un microcomputador sería deseable construir un orden de evaluación sobre dicha relación para ir desechando FOLLOWs que no se necesitarán más en el cálculo de los LookAheads.

Extendiendo la relación INCLUDES entre no-terminales,

Definición 3.12

Si A, B ∈ N,

A INCLUDES B si y solamente si existe un ítem de la forma B -> alfa.Abeta tal que lambda ∈ FIRST(beta).

Esta relación no depende del autómata sino de la gramática. Si

esta idea va a ser usada en el algoritmo 3.5, |alfa| > 0.

Definición 3.13

Si A ∈ N,

ITEMS(A) = U { U { B -> alfa.Abeta | lambda ∈ FIRST(beta) } | A INCLUDES B }.

Habiendo construido la relación INCLUDES para no-terminales sólo resta construir un orden de evaluación utilizando dicha relación. Recordando cómo sería un orden de evaluación usando un grafo de dependencia, tendríamos las siguientes restricciones sobre dicho orden, donde Orden(A) significa el orden de evaluación de los LAs para los ítems finales que tienen a A en su parte izquierda.

- Orden(B) < Orden(A) si y solamente si A INCLUDES* B y no B INCLUDES* A.
- Orden(B) = Orden(A) si y solamente si A INCLUDES* B y B INCLUDES* A. A y B tienen el mismo orden de evaluación si y sólo si A y B se encuentran en un ciclo de la relación INCLUDES.

De aquí encontramos que la relación INCLUDES entre las transiciones no-terminales de dos no-terminales A y B, no tiene necesidad de construirse, siempre y cuando ni A ni B tengan el mismo orden de evaluación. Si por ejemplo, existen (p,A) y (q,B) tales que (p,A) INCLUDES (q,B) no tenemos necesidad de construirla pues sabemos que Orden(B) < Orden(A) y por lo tanto los FOLLOWs de las transiciones no-terminales de B se han calculado. Simplemente, a FOLLOW(p,A) se le adiciona el FOLLOW(q,B).

Definición 3.14

Sea i la i-ésima posición en el orden de evaluación de los LAs,

Gi = { A | Orden(A) = i }.

El algoritmo para el cálculo de los LAs usando un orden de evaluación quedaría,

Algoritmo 3.6

```
    For Each i in 1..Max Orden de evaluación dO_Loop
       (*  Inicialización de los FOLLOWs y construcción de
           INCLUDES para los no terminales en este orden *)
       For Each A ∈ Gi
       dO_Loop
           For Each  (p,A) ¦ SUCC(p,A) está definida
           dO_Loop
               FOLLOW(p,A) := Ø          ;
               INCLUDES(p,A) := Ø        ;
           End_Loop                     ;
           For Each  B -> alfa.A beta ∈ ITEMS(A)
           dO_Loop
               For Each  (q,B) ¦ SUCC(q,B) está definida
               dO_Loop
                  p := SUCC(q,alfa)  ;
                  FOLLOW(p,A)  :=  FOLLOW(p,A) U FIRST(beta)  ;
                  dO_When  (  beta  =>*  lambda )
(*Línea 1*)          For Each  r ∈ PRED(p,alfa)
(*Línea 2*)       dO_Loop
(*Línea 3*)             For Each  C ->beta1.B'beta2 ∈ Kr
(*Línea 4*)                          donde B' I* B
(*Línea 5*)             dO_Loop
(*Línea 6*)                FOLLOW(p,A) := FOLLOW(p,A) U PATH(B',B) ;
(*Línea 7*)                dO_When ( lambda ∈ PATH(B',B) )
                              dO_When ( B' ∈ Gi )
                                INCLUDES(p,A):=INCLUDES(p,A) U
                                                { (r,B') }          ;
                              Else_dO
                                FOLLOW(p,A) := FOLLOW(p,A)   U
                                               FOLLOW(r,B')         ;
                              End_dO     ;
                           End_dO  ;
                        End_Loop  ;
                     End_Loop  ;
                  End_dO  ;
               End_Loop ;
           End_Loop  ;
       End_Loop ;
       (* Evaluación de la relación INCLUDES para los no-
          terminales presentes en este orden *)
       (*  Cálculo de los LAs para los ítems finales de los no-
           terminales envueltos en este orden *)
       For Each    A  ∈  Gi
       dO_Loop
          For Each A -> w. ∈ Cp  dO_Loop
              LA(q,A -> w.) := Ø      ;
             For Each q ∈ PRED(p,w), B -> beta1A beta2 - Kq
             dO_Loop
                 LA(q,A -> w.) := LA(q,A -> w.) U PATH(A',A) ;
                 dO_When ( lambda ∈ PATH(B',B) )
                     LA(q,A -> w.) := LA(q,A -> w.) U FOLLOW(q,A) ;
                 End_dO   ;
             End_Loop ;
```

```
            End_Loop    ;
        End_Loop    ;
    End_Loop    ;
```

Este algoritmo es válido para las dos mejoras (DeRemer y Pennello y Park), pues, para la mejora de DeRemer y Pennello es suficiente eliminar desde la (*Línea 1*) hasta la (Línea 7*) y en el caso de Park calcular el PATH de acuerdo con la definición. La mejora básica de este algoritmo sobre el de Park está en la generación del orden de evaluación. Recordando las estadísticas de DeRemer y Pennello, para gramáticas prácticas (PASCAL, MODULA, etc) el número de no-terminales envueltos en ciclos en la relación INCLUDES es muy pequeno (PASCAL 7 no-terminales), y en nuestro algoritmo sólo se crea la relación INCLUDES para los no-terminales presentes en el orden de evaluación. Normalmente, el número de transiciones no-terminales en un ciclo de la relación INCLUDES (para transiciones no terminales) tiene una cota superior en el número de arcos que aparecen en la relación INCLUDES (entre los no-terminales) presentes en el mismo orden de evaluación. Si sumamos todas las transiciones no-terminales de los no-terminales presentes en el mismo orden de evaluación y multiplicamos por la cota superior anterior obtendremos una cota superior (un poco exagerada) del número de arcos presentes en la relación INCLUDES para este algoritmo.

4. Resolviendo Conflictos

En esta sección esbozaremos algunas ideas que pueden emplearse en la construcción de algoritmos que ilustren el por qué se han producido ciertos conflictos en un autómata. La información puede utilizarse entonces para intentar resolver los conflictos, pero el éxito en esta labor dependerá de la astucia del usuario para desembarazarse de ellos.

Definición 4.1

Sea $q \in Q$ y $t \in T$,

q tiene un conflicto con t si y sólo si, SUCC(q,t) está definido y |Reduce(q,t)| > 0 y/o |Reduce(q,t)| > 1. Se llamará conflicto SHIFT-REDUCE si SUCC(q,t) está definido y Conflicto REDUCE-REDUCE si |Reduce(q,t)| > 1. Un conflicto a la vez puede ser SHIFT-REDUCE o REDUCE-REDUCE.

En el caso de un conflicto SHIFT-REDUCE debemos mostrar dos tipos de información, una debido a la componente del conflicto SUCC(q,t), que la denominaremos SHIFT-TRACE y otra debido a cada una de las producciones presentes en Reduce(q,t), que la denominaremos REDUCE TRACE.

Definición 4.2

Sea $q \in Q$, A -> alfa.beta $\in Cq$, decimos

CI(q,A -> alfa.beta) = { gamma ∈ V* | SUCC(i°, gamma alfa) = q }.

Sea q un estado que tenga un conflicto SHIFT-REDUCE con t. En qué consiste el SHIFT-TRACE ?. Si SUCC(q,t) está definido es porque existe por lo menos un ítem de la forma A -> alfa.tbeta ∈ Cq. Entonces, hay que justificar por qué A -> alfa.tbeta ∈ Cq, por lo tanto el SHIFT-TRACE es igual a CI(q,A -> alfa.tbeta) para cada ítem de la forma A -> alfa.tbeta ∈ Cq.

Sin embargo, CI es un conjunto demasiado grande (puede ser no finito), y a veces no es necesario presentar toda la información para entender el por qué del conflicto. Por ejemplo, presentando un elemento de CI que tenga la mínima longitud es suficiente (el contexto izquierdo más corto).

A continuación presentaremos un algoritmo que calcula el contexto izquierdo más corto, usando exclusivamente la función SUCC de un autómata ALR(0) cualquiera. Suponiendo que la numeración de los nodos del grafo asociado a dicho autómata se hubiera recorrido por extensión. La manera como se generó TP(G) en el algoritmo 3.2 nos permite obtener dicha numeración.

Definición 4.2

Sea alfa ∈ V* y p - 0,

MINPRED(p,alfa) = p si alfa es lambda
y
MINPRED(p,alfa) = MINPRED(MIN({ q|q ∈ PRED (p,Prim(alfa))}),
Rest(alfa)).

Definición 4.3

Sea p,q ∈ Q y X ∈ V,

ROTULO(q) = X si y sólo si SUCC(p,X) = q.

Teniendo las definiciones anteriores,

Algoritmo 4.1

```
Function   CIC(p,A -> alfa.beta) return V*  is

Begin
    CIC  := lambda          ;
    q := MINPRED(q,alfa)    ;
    WHILE  q  ≠ i°  dO_Loop
        CIC := Append(CIC,ROTULO(q)) ;
        q  :=  MINPRED(q,ROTULO(q))  ;
    End_Loop            ;
    return  (CIC)       ;
End CIC ;
```

Definición 4.4

Sean B,C ∈ N, t ∈ T, t ∈ PATH(B,C) y, por la definición de PATH(B,C), existen B0=B .., Bn=C y B0 -> B1beta1 ∈ P, B1 -> B2beta2 - P, ..., Bn-1 -> Bnbetan ∈ P y t ∈ FIRST(betan... beta1). Por último, existiría un i ∈ [0..n] tal que t ∉ FIRST($beta_n$...$beta_{i+1}$) y t ∈ FIRST($beta_i$).

A,

- betan betan-1 ..... betai+1 =>* lambda, los llamaremos los lambda PATH(B,C) de t para todo i que satisfaga la anterior condición ;
- betai =>* t betai', los llamaremos las FIRST-PATH(B,C) derivaciones de t para lambda PATH(B,C) de t ;
- B0 =>* B1 betan .... beta1 la PATH(B,C) derivación,
- B0 -> B1 beta1 la producción contribuyente.

los llamaremos PATH-TRACEs de t en PATH(B,C).

Definición 4.5

Sea q ∈ Q, A ∈ N, t ∈ T, decimos que,

beta =>* tbeta'

Es una FIRST derivación de t en FOLLOW(q,A) si y sólo si existe B -> alfa.Abeta - Kq, tal que t - FIRST(beta).

El siguiente algoritmo nos devolverá todos los B -> alfa.Abeta ∈Kq que tengan una FIRST derivación de t en el FOLLOW(q,A).

Algoritmo 4.2

```
Function  FIRST_deriv(q,A,t) return N x V* x V* Is

    Begin
        FIRST_deriv :=  Ø        ;
        For Each  B -> alfa.Abeta ∈ Kq ¦ t ∈ FIRST(beta)
        dO_Loop
           FIRST_deriv := FIRST_deriv U { B -> alfa.Abeta } ;
        End_Loop  ;
        return (FIRST_deriv)        ;
    End    ;
```

Definición 4.6

Sea q ∈ Q, A ∈ N, t ∈ T, y B -> alfa.Abeta ∈ Kq, tal que t ∉ FIRST(beta), lambda ∈ FIRST(beta) y B' I* B tal que t ∈ PATH(B',B),

A los PATH-TRACEs de t en PATH(B',B) los denominaremos las PATHs derivaciones de t en FOLLOW(q,A) y a B -> alfa.Abeta el ítem contribuyente.

Algoritmo 4.3

```
Function   PATH_deriv(q,A,t) return ( NxV*xV* )xN Is

Begin
    PATH_deriv := Ø          ;
    For Each  B -> alfa.Abeta ∈ Kq ¦ beta =>*
    dO_Loop
       PATH_deriv := PATH_deriv U
                     { (B -> alfa.Abeta,{B'¦ t ∈ PATH(B',B)}) }
    End_Loop  ;
    return (PATH_deriv)      ;
End ;
```

La anterior función halla los ítems contribuyentes con sus respectivos no terminales que a través del PATH adicionan un símbolo terminal cualquiera al FOLLOW de la debida transición no terminal. Esta información es necesaria para producir todas las PATH-derivaciones de un terminal en el FOLLOW de una transición no terminal cualquiera.

Definición 4.7

Sea q ∈ Q, A ∈ N, t ∈ T, decimos que, FOLLOW(q,A) incluye directamente a t, si y sólo si, hay una FIRST derivación de t en FOLLOW(q,A) o hay una PATH derivación de t en FOLLOW(q,A).

Definición 4.8

(p,A -> w) LOOKBACK (q,B) si y solamente si existe B tal que lambda ∈ PATH(B,A), q ∈ PRED(p,w).

Si t ∉ LA(p,A -> w), entonces o existe B ∈ N tal que t ∈ PATH(B,A) o existen r y C tales que
(p,A -> w) LOOKBACK (q,B) INCLUDES* (r,C)

y FOLLOW(r,C) incluye directamente a t.

Ambos casos pueden suceder, pero nos interesa justificar la ocurrencia de al menos uno de ellos. Al primer caso, los vamos a llamar una inclusión directa de t en LA(p,A -> w); y al segundo, una inclusión indirecta de t en LA(p,A -> w). Las justificaciones de t - LA(p,A -> w) son todas las derivaciones de las inclusiones directas e indirectas y según las definiciones anteriores, se están abarcando todos los casos. Como siempre, para propósitos prácticos solo es deseable mostrar una de tales derivaciones: La primera que se encuentre según el método de búsqueda que sea empleado. No daremos algoritmos completos para la obtención directa de esta información, sino que nuestra intención es mostrar ideas generales para su desarrollo así como los algoritmos básicos para obtenerla.

5. Conclusiones

Las ideas desarrolladas en este trabajo, junto con las tomadas de los artículos descritos en la introducción y en la bibliografía, han sido hechas realidad en un sistema generador de analizadores sintácticos que ha sido usado como material de laboratorio en el curso de compiladores de la Universidad de los Andes (Bogotá, Colombia) con bastante éxito. La principal virtud del sistema es el tiempo de respuesta, debido en parte a la reducción de cálculos obtenida gracias al algoritmo 3.6.

El sistema, como se encuentra actualmente, recibe como entrada una gramática en forma de BNF o BNF-extendida y produce como salida un programa fuente manejador de la tabla de parsing, escrito en TURBOPASCAL, una descripción completa del autómata y las tablas, así como información para ayudar a resolver los conflictos.

Debido al excelente tiempo de respuesta y a la cantidad de información que proporciona, se ha convertido en una herramienta didáctica muy apreciada por los estudiantes, sobre todo por el hecho de ser utilizable sobre microcomputadores lo que le da una flexibilidad de acceso difícil de obtener sobre un computador grande (máquinas en las que tradicionalmente se ha implementado este tipo de software). Por otro lado, además de haber sido usado como material de laboratorio, el compiler-compiler ha sido probado para diferentes gramáticas "reales", entre las cuales tenemos: MODULA, TURBOPASCAL, un subconjunto de ADA (no hay problemas en la generación de ADA), una serie de gramáticas para generación de reportes y la gramática de PASCAL ANSI, con resultados satisfactorios, demostrando que el sistema es útil en un ambiente de producción real y es algo más que una facilidad didáctica.

En el estado actual, los fuentes están siendo pasados a C para obtener un mejor desempeño, aunque el tiempo de respuesta es excelente (sobre todo teniendo en cuenta que se utiliza en microcomputadores standard de 16 bits). Adicionalmente, se ha pensado agregarle otros módulos que comprenden: generador autómatico de tablas de símbolos, un generador automático de analizadores léxicos, un compactador de tablas esparcidas, un sencillo evaluador de atributos, un algoritmo general para generación de código para expresiones y un lenguaje para la especificación de lenguajes de máquina.

Referencias

[AHO77] Aho A, Ullman J.
"Principles of Compiler Design"
Addison-Wesley, Reading Mass., 1977.

[DeR82] DeRemer F, Pennello T.J.
"Efficient Computation of LALR(1) LookAhead Sets"
ACM TOPLS Vol. 4 No. 4, octubre 1982

[MON85] Montes A.
"Diseño e Implementación de un Generador de Compiladores"
Proyecto de Grado, Dpto. de Sistemas, UNIANDES, Bogotá, 1985.

[PAR85] Park J, Choe K, Chang C.
"A New Analysis of LALR Formalisms"
ACM TOPLS Vol. 7 No. 1, enero 1985.

LOS LENGUAJES FUNCIONALES - NUESTRA EXPERIENCIA CON KRC

Miguel Santana - Fabienne Carrier
Institut IMAG - *Francia*
Eva Rodríguez
Universidad Central de Venezuela - *Venezuela*

1. INTRODUCCION

La importancia actual de los lenguajes funcionales puede ser atribuída en gran parte a la crisis del software, que ha hecho pensar en estos lenguajes como una alternativa a los lenguajes de programación clásicos.

Efectivamente, debido a su poder de expresión, los lenguajes funcionales corresponden bien a las necesidades de las aplicaciones recientes de la Informática: sistemas expertos, comunicación hombre-máquina, ingeniería del software, etc. Por otro lado, son capaces de explotar las posibilidades de cálculo paralelo de las arquitecturas hardware modernas, gracias a su paralelismo intrínseco.

Sin embargo, estas características no han bastado para que estos lenguajes se impongan. Efectivamente, ciertos defectos les son atribuídos por los programadores habituados a los lenguajes clásicos; se les reprocha en particular su ineficiencia en los computadores actuales así como la dificultad para escribir una aplicación real.

El objetivo de este artículo es de estudiar la adecuación de los lenguajes funcionales a la expresión de problemas informáticos clásicos; para ésto, nos servimos de un lenguaje funcional moderno sobre el cual hemos trabajado durante los dos últimos años: KRC (Kent Recursive Calculator). Nuestro trabajo forma parte de un proyecto más ambicioso [Briat 85], cuyo objetivo final es la realización de un sistema de producción de software, en base a un lenguaje de la quinta generación y a una arquitectura paralela (16 a 1024 procesadores).

La sección 2 de este artículo describe de manera informal el lenguaje KRC, ilustrándolo con múltiples ejemplos. La sección 3 presenta una evaluación de KRC y, a través de ella, de los lenguajes funcionales en general; dos aspectos constituyen la parte más importante de nuestro análisis: el poder de expresión de estos lenguajes y su aplicación eventual a un contexto real de producción de software. Luego, describimos en la sección 4 la implementación de KRC que realizamos para llevar a cabo nuestras experiencias. Finalmente, presentamos en la sección 5 las características que nos parece indispensable agregar a los lenguajes funcionales para volverlos *utilisables*, antes de concluir sobre el interés de nuestro trabajo y sobre las perspectivas de desarrollo futuro.

2. DESCRIPCION DEL LENGUAJE KRC.

KRC es un lenguaje funcional puro, en el cual no existen las nociones de efectos secundarios (la afectación, en particular) ni de flujo de control; es un lenguaje simple, basado en las ecuaciones recursivas de orden superior. Fue diseñado en la Universidad de Kent con un objetivo preciso: la enseñanza de la programación funcional [Turner 81, Turner 82].

KRC es además une lenguaje interactivo; efectivamente, la definición de ecuaciones así como la evaluación de expresiones son efectuadas dinámicamente, tal como sucede en los sistemas LISP. Por otro lado, el orden de definición de las ecuaciones no tiene ninguna importancia; sólo es necesario que una ecuación ya haya sido definida cuando la evaluación de la expresión en curso lo requiere.

Un programa KRC es un conjunto de ecuaciones; estas ecuaciones constituyen una definición matemática de los objetos usados por el programa y de las relaciones existentes entre estos objetos. KRC dispone de dos tipos de ecuaciones: las definiciones y las expresiones simples. Las primeras asocian un nombre a un valor o a una función, que puede luego ser usado en otras ecuaciones. Las expresiones simples son evaluadas inmediatamente y el resultado impreso en la pantalla; estas expresiones corresponden por consiguiente a la ejecución de un programa.

Los diferentes aspectos del lenguaje son presentados en los párrafos que siguen. Nuestra presentación es voluntariamente concisa y se apoya en ejemplos simples para ilustrar cada aspecto.

2.1. OBJETOS DEL LENGUAJE.

KRC dispone de dos clases de objetos: elementales y estructurados. Estos últimos son definidos por el usuario gracias a ciertas construcciones del lenguaje y son de dos tipos: listas y funciones. Cabe indicar que un objeto estructurado puede ser manipulado como cualquier otro objeto y, en particular, ser pasado como parámetro o devuelto como resultado de una función.

2.1.1. OBJETOS ELEMENTALES.

Cuatro tipos de objetos predefinidos existen en KRC: enteros, reales, cadenas de caracteres y lógicos. Todas las operaciones aritméticas, lógicas y de comparación usuales pueden ser usadas con estos objetos. Las expresiones que siguen constituyen un ejemplo de lo que puede ser escrito:

```
-100 + 5654
1.3 / 12e2 * -8.0
"string1" < "string2"  |  15e3 >= 7685
```

2.1.2. LISTAS.

Una lista KRC es un conjunto ordenado de objetos. Estos objetos, llamados elementos de la lista, pueden ser de cualquier tipo; el orden entre los elementos es definido por sus posiciones respectivas dentro de la lista. Una lista es definida por enumeración de sus elementos:

```
[]      [1,2,3,4,5]        ["a","b"]
[ "David", [15,"Janvier",85], true]
```

Un intervalo de enteros puede ser definido usando una forma sintáctica más agradable: `[-2..2]` (equivalente a `[-2,-1,0,1,2]`). Es igualmente posible definir un intervalo infinito de enteros: `[1..]` (enteros positivos).

Una gran cantidad de operadores y de funciones de manipulación de listas existen en

KRC, entre ellos:

- adición de un elemento al comienzo de una lista (*cons*):
  `"a": ["e","i","o"]  0: [1..]`
- acceso al elemento n^{o} i de una lista (indexación):
  `index( ["a","b","c"], 2 )` cuyo valor es `"b"`.
- longitud de una lista:
  `# [1..5]` cuyo resultado es 5.
- concatenación de dos listas:
  `["a","b","c"] ++ ["d","e"]` igual a `["a","b","c","d","e"]`
- diferencia de dos listas:
  `[1..] -- [1..5]` equivalente a `[6..]`
- cabeza y cola de una lista:
  `hd( ["e","i","o"] )` cuyo valor es `"e"`
  `tl( ["e","i","o"] )` cuyo valor es `["i","o"]`

etc.

Además, dos listas pueden ser comparadas usando los operadores de igualdad y de desigualdad usuales:

```
[1,2,3,4,5]   =  [1..5]
["e","i","o"] <> tl( ["e","i","o"] )
```

La construcción de una lista es efectuada según un mecanismo de evaluación perezosa [Friedman 76]. La construcción efectiva es por consiguiente retardada hasta el primer acceso a la lista, momento en el cual son construídos los elementos de la lista implicados en ese acceso y sólo ellos; por esta razón, una lista infinita no conduce necesariamente a una iteración infinita.

2.1.3. FUNCIONES

Una función es definida por una o varias ecuaciones. Cada ecuación está constituída de un encabezamiento, que define su nombre y sus parámetros, y de un cuerpo, que define el valor de la ecuación. Todas las ecuaciones de una misma función deben tener el mismo nombre y el mismo número de parámetros; si este último es igual a cero, se trata de la definición de una constante.

La notación usada para designar la aplicación difiere ligeramente de aquella que usa Turner. Efectivamente, hemos preferido que los parámetros sean especificados entre paréntesis y separados por comas, en vez de usar la yuxtaposición simple, como Turner. Esta notación otorga una mejor lisibilidad y sobre todo, facilita la tarea de análisis sintáctico (ambiguedades, recuperación de errores). Los ejemplos que siguen ilustran las diferentes posibilidades:

```
% Suma de los n primeros enteros %
def suma(n) = ((n+1)*n) / 2.
suma(4).   % suma de los 4 primeros enteros %
```

```
% Días de la semana %
def laborables = ["lunes","martes","miércoles","jueves","viernes"].
def weekend    = ["sábado","domingo"].
def semana     = laborables ++ weekend.
index(semana,3).   % tercer día de la semana %
```

Como una función puede tener varias ecuaciones, el usuario puede especificar las condiciones de aplicación de cada ecuación, usando dos tipos de construcciones disponibles en KRC:

a) Las guardas, expresiones lógicas asociadas al cuerpo de una ecuación. Estas guardas permiten de escoger entre las diferentes expresiones que componen el cuerpo de una ecuación; su semántica es similar a la del *cond* LISP: una expresión es evaluada únicamente si su guarda es verdadera. El orden de evaluación de estas guardas no es impuesto por el lenguaje; sin embargo, una sola expresión debe ser seleccionada y ejecutada. La última expresión de la ecuación no debe tener una guarda, lo que podría conducir a una aplicación indefinida. Ejemplo:

```
% El más grande común divisor %
def mgcd(i,j) = i,          i=j,
                mgcd(i-j,j), i>j,
                mgcd(i,j-i).
```

b) Los filtros, que corresponden a restricciones sobre los valores de los parámetros de la ecuación. Estas restricciones son definidas mediante una especie de *pattern matching*; un parámetro puede ser restringido a un valor dado o a una forma determinada de lista. Todos los filtros de una ecuación tienen que ser respetados para que la ecuación pueda ser aplicada. Ejemplos:

```
% Factorial %
def fact(0) = 1;
    fact(1) = 1;
    fact(n) = n*fact(n-1).

% Suma de los elementos de una lista %
def suma([])          = 0;
    suma(primero:resto) = primero + suma(resto).

% Multiplicación de números complejos %
def mult([r1,i1],[r2,i2]) = [r1*r2-i1*i2,r1*i2+i1*r2].
```

Ninguna limitación existe sobre el tipo de los parámetros; una función puede perfectamente recibir o retornar otra función. Los ejemplos que siguen ilustran esta posibilidad:

```
% Aplicación de una función a los elementos de una lista %
def map(f,[])     = [];
    map(f,head:tail) = f(head): map(f,tail).

% Identificación de los comandos de un editor. El resultado
  es el nombre de la función que efectuará el tratamiento %
def comando(clave) = insert, clave="i",
                     delete, clave="d",
```

```
...
unknown.
```

2.2. CONJUNTOS.

Otra facilidad disponible en KRC para la construcción de listas son los conjuntos. Esta facilidad permite expresar las nociones de conjunto y de cuantificación, sin tener que escribir explícitamente una recursión. Es necesario precisar que el resultado obtenido con esta construcción no es verdaderamente un conjunto sino una lista; un valor puede así aparecer varias veces en este resultado.

Los conjuntos KRC constituyen una notación muy poderosa; así, por ejemplo, el conjunto de enteros pares positivos puede ser definido de una manera muy simple:

```
{ i; i <- [1..], i mod 2 = 0 }
```

Como se puede ver en este ejemplo, la definición de un conjunto comprende tres partes:
- una expresión de definición i, que especifica los elementos del conjunto,
- un generador i <- [1..], que precisa el dominio de valores en el cual está definida la variable i,
- una guarda i mod 2 = 0, que define la condición que deben respectar los valores generados para pertenecer al conjunto.

Ejemplos:

```
% Otra definición de la función map %
def map(f,lista) = { f(elemento); elemento <- lista }

% Conjunto de números primos (método de Eratosteno) %
def primos      = filtro( [2..] ).
def filtro(p:r) = p: filtro( {e; e<-r, (e div p) > 0} ).
```

La definición de un conjunto puede contener varios generadores. El orden de definición de estos generadores es importante porque es él quien define la visibilidad de la variable asociada; efectivamente, una variable sólo puede ser usada en un generador que no es el suyo, si ya ha sido definida por un generador anterior. En virtud de esta característica, la guarda de un generador puede constituir una condición global, si hace intervenir variables de generadores precedentes. Cabe indicar que la expresión de definición del conjunto puede usar cualquier variable. Ejemplo:

```
% Parejas de números pares cuya suma es igual a 10 %
{ [i,j]; i <- [1..10], i mod 2 = 0;
         j <- [1..10], j mod 2 = 0  &  i+j = 10 }
```

Para terminar, es importante indicar que un conjunto puede ser usado en lugar de una lista, sin restricción alguna.

2.3. EJEMPLOS.

El objetivo de este párrafo es de ilustrar el poder de expresión de KRC. Para ello, presentamos la versión KRC de algoritmos clásicos, que tienen la doble virtud de ser ampliamente conocidos y de escritura relativamente sucinta.

2.3.1. FUNCIONES CLASICAS.

```
% FUNCION DE ACKERMANN %

def ackermann( 0, n ) = n + 1;
    ackermann( m, 0 ) = ackermann( m-1, 1 );
    ackermann( m, n ) = ackermann( m-1, ackermann(m,n-1) ).

% NUMEROS DE FIBONACCI %

% Definición standard %
def fib( 1 ) = 1;
    fib( 2 ) = 1;
    fib( n ) = fib( n-1 ) + fib( n-2 ).

% Versión optimizada: cada número es calculado una vez %
def fib'( 1 ) = 1;
    fib'( 2 ) = 1;
    fib'( n ) = index( n-1, fiblist ) + index( n-2, fiblist ).
def fiblist   = fibl( 1 ).
def fibl( i ) = fib'( i ): fibl( i+1 ).
```

2.3.2. FUNCIONES DE ORDEN SUPERIOR. GENERICIDAD.

```
% DOBLE APLICACION DE UNA FUNCION %

def twice( f, val ) = f( f(val) ).

def sqr( i ) = i * i.
def power4   = twice( sqr ).

% APLICACION DE UNA FUNCION CUALQUIERA A UNE LISTA %

def fold( f, init, [] )  = init;
    fold( f, init, p:r ) = f( p, fold(f,init,r) ).

def add( op1, op2 ) = op1 + op2.
def sum             = fold( add, 0 ).
```

2.3.3. APLICACION DE LOS CONJUNTOS.

```
% PRODUCTO CARTESIANO %

def cartesiano( l1, l2 ) = { [i,j]; i <- l1; j <- l2 }.

% NUMEROS PRIMOS %

def primos =
       { primo; primo <- [2..50],
         { divisor; divisor <- [ 2..round( sqrt(primo+1) ) ],
                    primo mod divisor = 0 }
         =
         [] }.
```

2.3.4. QUICKSORT.

```
% SEGMENTACION + FUSION %

def quicksort( [p] )     = [];
    quicksort( i:resto ) =
       fusion( i, segmentar(i,resto,[],[]) ).

def fusion( pivot, [parte1,parte2] ) =
       quicksort( parte1 ) ++ pivot: quicksort( parte2 ).

def segmentar( pivot, [], parte1, parte2 ) =
       [ parte1, parte2 ];
    segmentar( pivot, i:resto, parte1, parte2 ) =
       segmentar( pivot, resto, i:parte1, parte2 ), i <= pivot,
       segmentar( pivot, resto, parte1, i:parte2 ).

% DEFINICION POR CONJUNTOS %

def quicksort'( [] )  = [];
    quicksort'( p:r ) = quicksort'( { v; v<-r, v<=p } )
                        ++
                        p: quicksort'( { v; v<-r, v>p } ).
```

2.3.5. PROBLEMA DE LAS 8 REINAS.

```
% SOLUCION CLASICA %

def OchoReinas (0) = [ [] ];
    OchoReinas (n) = {  ListaReinas ++ [reina];
                        reina <- [1..8];
                        ListaReinas <- OchoReinas (n-1),
                        correcta (reina, ListaReinas) }.
```

```
def correcta (reina, lista) =
       and ( { ~test (reina,lista,i); i <- [1..#lista] } ).

def test (reina, lista, i) =
       reina = index (i,lista)
       |  abs (reina - index (i,lista)) = (#lista-i)+1.
```

3. EVALUACION DE KRC.

A primera vista, KRC constituye un lenguaje sumamente atractivo y con un gran poder de expresión, como lo muestran los ejemplos presentados. Sin embargo, un análisis más detallado incita a preguntarse si podría ser usado con éxito en la realización de una aplicación real. Para tratar de responder a esta pregunta y a otras similares, nos pareció necesario efectuar una evaluación de este lenguaje, tratando de separar claramente sus virtudes y sus límites; fue así como decidimos escribir en KRC un cierto número de aplicaciones, pertenecientes en su mayor parte al área del Software de base y de la Ingeniería del Software. Los párrafos que siguen presentan las conclusiones más importantes de esta experiencia; para mayores precisiones, el lector puede consultar [Carrier 85].

3.1. PODER DE EXPRESION.

Dos aspectos de KRC nos parecen de un gran interés: su notación ecuacional y su mecanismo de filtrado de parámetros; estos dos aspectos se amoldan perfectamente a la expresión de una cantidad apreciable de problemas. Por este motivo, KRC ofrece un nivel de abstracción bastante elevado, cercano de la especificación; efectivamente, la definición KRC de un problema recuerda muchas veces la formulación matemática del mismo. Este nivel de abstracción permite al usuario de expresar su algoritmo sin preocuparse de los detalles relativos a su implementación: representación de sus datos, administración de la memoria, flujo de ejecución, etc. (nociones indispensables en un lenguaje imperativo).

El mecanismo de filtrado ofrece dos ventajas importantes, que facilitan enormemente la labor del programador. Por un lado, permite distinguir los diferentes casos de una función según la structura de los parámetros; por otro lado, reduce el uso de las funciones de acceso, mejorando la lisibilidad del programa y reduciendo los errores (muchas veces delicados) correspondientes a este tipo de acceso. Las dos definiciones siguientes ilustran estos aspectos:

```
def suma3 (triple)  =
          hd(triple) + hd(tl(triple)) + hd(tl(tl(triple))).

def suma3 ([a,b,c]) = a + b + c.
```

Nos han igualmente seducido por su poder de expression los conjuntos KRC. Efectivamente, estos conjuntos permiten especificar de una manera simple y concisa una noción importante de la programación: la aplicación de un mismo tratamiento a una colección de datos (la iteración en los lenguajes clásicos); esta posibilidad es además enriquecida por un mecanismo de filtrado, que permite seleccionar el sub-conjunto de la colección al cual se quiere aplicar el tratamiento. Por esta razón, los conjuntos KRC se adaptan muy bien a la

expresión de una gama importante de algoritmos, que se pueden resumir en términos de *el conjunto de objetos que verifican tal propiedad.*

Otro aspecto interesante de KRC es la semántica no estricta de sus funciones, lo que les permite retornar un resultado inclusive cuando ciertos parámetros no son definidos; en particular, la evaluación de un parámetro que no es usado y que conduciría a una evaluación infinita no es jamás efectuada. Esta propiedad garantiza el término de la ejecución de una función cuando existe por lo menos una solución de ésta. Este mismo principio es aplicado a las listas, lo cual otorga la posibilidad de definir estructuras de datos potencialmente infinitas.

Finalmente, ciertas propiedades de KRC, y de los lenguajes funcionales en general, nos parecen igualmente de suma importancia. La más importante de ellas es la transparencia de las funciones: dado un mismo conjunto de parámetros, una función siempre da el mismo resultado; esta propiedad está ligada a la ausencia de afectación en el lenguaje y al hecho que una variable representa un valor y no un objeto, sujeto a modificaciones. La importancia de esta propiedad es doble:

a) Posibilidad de efectuar diversos tratamientos formales: prueba de igualdad entre dos funciones, transformaciones, prueba de programas, etc. Esta posibilidad se ve reforzada por la solidez de las bases teóricas sobre las cuales reposan estos lenguajes: lambda-cálculo, lógica combinatoria, etc.

b) Posibilidad de efectuar una implementación paralela. Diferentes partes de un programa pueden efectivamente ser efectuadas en paralelo: evaluación de los argumentos de una función, construcción de los elementos de una lista, etc.

En conlusión, podemos decir que KRC constituye un lenguaje poderoso, que ofrece un poder de expresión y un nivel de abstracción bastante elevados. Sin embargo, ciertos aspectos de KRC nos parecen contrarios a un uso real del lenguaje; el análisis de estos inconvenientes constituye el objeto del párrafo siguiente.

3.2. APLICACION A UN CONTEXTO DE PRODUCCION.

La producción industrial de software tiene sus imperativos y exigencias en cuanto a la calidad y cualidades de las herramientas de desarrollo; por esta razón, un lenguaje como KRC sólo sería utilizado en este contexto si respeta todas estas condiciones. Las líneas que siguen tratan de identificar, por un lado, las características de KRC que se adaptan a este contexto y, por otro lado, aquellas que son contrarias a él o que son simplemente ausentes.

Una primera característica importante de KRC es la notación clara y natural que utiliza, lo que le da una excelente lisibilidad.. Esta característica constituye una ventaja importante en relación a otros lenguajes funcionales; no estamos frente a la gran cantidad de paréntesis de LISP o de caracteres especiales de FP. Este aspecto reviste una gran importancia en un contexto de producción de software, debido a la evolución constante de éstos (mantenimiento y nuevas versiones).

El nivel de abstracción de KRC, bastante cercano de la especificación, constituye otra ventaja importante de este lenguaje. Efectivamente, esta similitud nos hace pensar en una reducción del ciclo de producción de un software y, por consiguiente, de su costo.

Sin embargo, estas virtudes aliadas a su poder de expresión no confieren a KRC la categoría de *lenguaje de producción*. Muchas características, usadas intensivamente en la producción de software, son completamente ausentes o demasiado simples en KRC y, generalizando, en los lenguajes funcionales:

a) La pobreza de las estructuras de datos disponibles: en la mayoría de casos, la lista constituye la única estructura de datos existente. Esta pobreza dificulta considerablemente la programación de ciertas aplicaciones; en particular, un gran número de parámetros son a menudo necesarios para transmitir a una función las informaciones que necesita.

b) La manipulación eficaz de estructuras de datos. Estas operaciones han sido definidas por lo general en torno a la noción de variable: agrupamiento de variables, lectura o modificación de una de estas variables, etc. Esta definición es totalmente incompatible con los conceptos de base de los lenguajes funcionales, en los cuales una estructura de datos es considerada como un todo. Esta característica es una fuente importante de ineficiencia, tanto desde el punto de vista ocupación de memoria como de tiempo de ejecución, pues, toda modificación implica la reconstrucción completa de la estructura de datos.
Para resolver este problema, dos soluciones son posibles. La primera consiste en transgredir los principios de estos lenguajes, autorizando la manipulación de variables; es la solución adoptada por LISP o ML. La segunda solución consiste en efectuar una administración más fina de la memoria, encargada entre otras cosas de efectuar las modificaciones de estructuras de datos de una manera más eficiente [O'Donnell 85]; esta solución ha sido muy poco estudiada y merecería una mayor atención.

c) Las entradas/salidas. La noción de entrada/salida está en conflicto total con las propiedades de base de los lenguajes funcionales; efectivamente, las entradas/salidas han sido definidas como si se tratara de afectaciones a, o de, una variable externa (archivo, periférico). Una consecuencia de esta definición es la falta de transparencia de toda función de entrada/salida; así, por ejemplo, la función *read* no retorna siempre el mismo valor sino los elementos consecutivos de un archivo. Otra consecuencia es la incompatibilidad entre la secuencialidad de las entradas/salidas y la ausencia de orden de evaluación de los lenguajes funcionales, que puede conducir a resultados sorprendentes; así, por ejemplo, es difícil decir en que orden serán ejecutadas las llamadas de la función *read* en la expresión *f(read(), read())*.
El tratamiento de las entradas/salidas nunca ha sido completamente resuelto en los lenguajes funcionales. Una idea que comienza a popularizarse consiste en tratar un archivo como un flujo de datos (*stream*), cuya evaluación es efectuada por intermedio del mecanismo de evaluación perezosa. Esta idea parece sumamente interesante pero necesita ser desarrollada un poco más.

Otros problemas son específicos a KRC. Entre ellos podemos citar:

a) Imposibilidad de definir variables locales a una función. Esta carencia, aunada a la imposibilidad de definir el cuerpo de función como una secuencia de expresiones, hacen que la programación en KRC sea bastante enredada cuando una función tiene que calcular un valor intermediario. Ciertos problemas simples se convierten de esta manera en verdaderos rompecabezas, en los que el programador se ve obligado de introducir funciones intermediarias que *simulan* estas posibilidades. La introducción de una construcción de tipo *let* permitiría de resolver este problema.

b) Ausencia de funciones anónimas (el equivalente de las expressiones lambda de LISP). Esta construcción es particularmente necesaria cuando uno desea pasar como parámetro o retornar como resultado una función.

c) Ambiguedades e imprecisiones en la definición del lenguaje: exclusividad entre las guardas de una función, orden de evaluación de los generadores de un conjunto, etc. Estos problemas son sin embargo de poca importancia y pueden ser fácilmente resueltos.

d) Noción de tipo. Ningún mecanismo de elaboración ni de verificación de tipos existe en KRC. Cuando uno sabe que muchos errores de programación son detectados por estos mecanismos, sólo puede lamentarse de esta ausencia.

Como se puede observar, la mayor parte de las carencias propias de KRC son menores con respecto a las que se puede imputar a los lenguajes funcionales en general. Una buena implementación del lenguaje puede resolver estos pequeños problemas, a partir de una definición precisa y de unas cuantas extensiones al lenguaje.

Todos los problemas señalados no deben llevarnos a rechazar estos lenguajes sino, por el contrario, a tratar de mejorarlos y de encontrar las técnicas y métodos apropiados para imponerlos. La sección que sigue presenta los puntos que, en nuestra opinión, deben ser estudiados y desarrollados en prioridad para lograr este objetivo.

4. NUESTRA IMPLEMENTACION DE KRC.

La realización de nuestro proyecto necesitaba una experimentación real, para poder obtener resultados significativos. Como el único sistema KRC existente no podía ser instalado en nuestro laboratorio por razones materiales, decidimos efectuar nuestra propia implementación de KRC. Esta decisión se vió corroborada por nuestro deseo de experimentar, por un lado, con las técnicas de implementación de los lenguajes funcionales y, por otro lado, con el lenguaje mismo.

Nuestra implementación fue realizada en un VAX-780, usando Franzlisp como lenguaje de desarrollo. Esta sección presenta los diferentes aspectos de esta implementación; una parte importante es dedicada a la técnica de evaluación utilizada, por la importancia que reviste en cuanto a la eficiencia del sistema.

4.1. TECNICA DE EVALUACION.

La semántica de KRC impone un orden normal de evaluación, tanto en lo que concierne las listas (evaluación perezosa) como en la transmisión de parámetros (pasaje por nombre o por necesidad). La técnica de los combinadores [Turner 79] se adapta muy bien a un tal lenguaje, pues ofrece directamente un orden normal de evaluación. En cambio, la lambda-reducción corresponde a un orden aplicativo, el costo de su adaptación a un orden normal resultando relativamente elevado.

Esta razón, aunada a nuestra deseo de experimentar con este nuevo método de evaluación, nos condujo a escoger la técnica de los combinadores. Esta técnica se basa en ciertos resultados

de la lógica combinatoria y especialmente en la demostración [Curry 58] de que las variables no son necesarias al cálculo de una ecuación y pueden ser reemplazadas por un cierto tipo de constantes, llamadas combinadores. Un algoritmo de transformación (abstracción de variables), capaz de transformar cualquier expresión en su equivalente sin variables, fue igualmente propuesto. La evaluación de estas expresiones puede enseguida ser efectuada aplicando un conjunto de reglas de reducción, que definen la semántica de los combinadores usados.

La técnica de los combinadores fue propuesta por Turner como una alternativa a la técnica clásica de evaluación de los lenguajes funcionales [Landin 64, Baker 78]. Ha sido usada en la implementación del lenguaje SASL así como en la realización de la máquina SKIM [Clarke 80, Stoye 84]; en ambos casos, los resultados obtenidos han sido sumamente interesantes tanto desde el punto de vista de la eficiencia como de la sencillez de la realización. Los párrafos que siguen presentan los diferentes aspectos de esta técnica.

4.1.1. DESCRIPCION GENERAL.

En el método de Turner, todo programa es traducido en un conjunto de expresiones, compuestas exclusivamente de combinadores, de constantes y de aplicaciones de funciones. La evaluación de una expresión consiste en reducir su forma combinatoria en un valor elemental.

Los dos combinadores de base de Curry son usados:

```
S f g x = f x (g x)
K x y   = x
```

Según las conclusiones de Curry, estos combinadores son suficientes para calcular cualquier expresión funcional. Sin embargo, otros combinadores son necesarios esencialmente por razones prácticas; efectivamente, estos nuevos combinadores reducen considerablemente la talla de las expresiones combinatorias y disminuyen al mismo tiempo el número de pasos de reducción. Se trata en realidad de optimizaciones de ciertas formas de S y de K:

```
I x     = x
B f g x = f (g x)
C f g x = f x g
```

Así, por ejemplo, la expresión combinatoria

```
S (C (K +) ((* 4) 2)) (K 5)
```

equivalente a

```
4*2 + 5    o    ((+ ((* 4) 2)) 5)  (forma currificada)
```

sería evaluada por la serie de reducciones siguiente:

```
S (C (K +) ((* 4) 2)) (K 5) nil
=> (C (K +) ((* 4) 2) nil) (K 5 nil)    REDUCCION S
=> (K + nil ((* 4) 2)) (K 5 nil)        REDUCCION C
=> (+ ((* 4) 2)) (K 5 nil)              REDUCCION K
=> (+ 8) (K 5 nil)                      EVALUACION *
=> (+ 8) 5                              REDUCCION K
=> 13                                   EVALUACION +
```

El método de Turner puede ser descompuesto en dos etapas, descritas a continuación.

4.1.2. ABSTRACCION DE VARIABLES.

Esta etapa consiste a suprimir las variables usadas por la expresión tratada. La supresión es efectuada por un algoritmo conocido con el nombre de abstracción de variables; la operación de abstracción puede ser definida de la manera siguiente:

Sean la expresión E y la variable x. La abstracción de E respecto de x, cuya notación es [x] E, es una expresión que no contiene ninguna ocurrencia de x y que respeta la ecuación:

([x] E) x = E

La abstracción es por lo tanto la operación inversa de la aplicación.

Esta operación puede ser usada para suprimir los parámetros de una función, en la expresión que constituye su cuerpo. De esta manera, la ligazón entre parámetros efectivos y parámetros formales es inútil, lo cual conduce a una mayor eficiencia respecto del lambda-cálculo.

El algoritmo de abstracción es definido por un conjunto de reglas de transformación, que permiten traducir una expression lambda (tipo LISP) en una expresión combinatoria. La expresión inicial tiene que estar en una forma currificada: toda aplicación de función debe tener un solo argumento. Las reglas de transformación son las siguientes:

```
[x] (E1 E2) => S ([x] E1) ([x] E2)
[x] x       => I
[x] y       => K y
```

donde x es la variable que se quiere abstraer e y una constante o una variable; la aplicación es considerada asociativa por la izquierda.

Además, dos reglas de optimización son usadas por el algoritmo:

```
S (K E1) (K E2) => K (E1 E2)
S (K E1) I      => E1
```

Finalmente, los combinadores B y C constituyen dos casos particulares del combinador S, en los cuales la variable x está ausente:

```
S (K E1) E2 => B E1 E2
S E1 (K E2) => C E1 E2
```

El conjunto de reglas debe ser aplicado en el orden definido para obtener un mejor resultado.

Otras reglas de optimización fueron definidas como consecuencia de una observación simple: la aplicación repetitiva de la abstracción provoca la aparición frecuente de ciertas expresiones tipo. Otras formas más simples y eficientes de estas expresiones pueden ser usadas:

```
S (B EK E1) E2 => S' EK E1 E2
B (EK E1) E2   => B' EK E1 E2
C (B EK E1) E2 => C' EK E1 E2
```

donde EK es una expresión constante y E1, E2 son cualquier tipo de expresiones. Estas optimizaciones son aún más interesantes si el número de variables a abstraer es importante.

El algoritmo de abstracción tiene que ser aplicado tantas veces como variables hay. Así, por ejemplo, la abstracción de la función:

```
f(x, y, z) = E
```

corresponderá a la expresión:

```
[z] ([y] ([x] E))
```

La abstracción de variables puede ser extendida al nombre de la función e inclusive a las variables y funciones globales.

La expresión obtenida puede ser representada como una forma arborescente, en la cual todos los nodos internos corresponden a la operación de aplicación y las hojas a una constante, un combinador o un nombre de función.

4.1.3. REDUCCION DE UNA EXPRESION.

La etapa de reducción corresponde al cálculo del valor de una expresión. Este cálculo es realizado por un algoritmo de transformación de grafo; efectivamente, la forma arborescente de la expresión es transformada por aplicación sucesiva de las reglas de reducción que definen los combinadores. Este algoritmo implementa de manera natural una evaluación perezosa.

La evaluación es efectuada según un orden normal de reducción. Una etapa de evaluación consiste a transformar la expresión mediante la regla de reducción asociada a la hoja ubicada más hacia la izquierda; la reducción es efectuada hasta obtener un valor elemental.

En el orden normal de reducción, los parámetros son transmitidos según el esquema de llamada por nombre; la ventaja de este esquema es que garantiza que la ejecución se termina, siempre y cuando sea posible. En la técnica de los combinadores, este esquema es reemplazado por la llamada por necesidad, esquema equivalente pero mucho más eficiente. Efectivamente, en este esquema los parámetros efectivos son únicamente apuntadores hacia la verdadera expresión, lo cual permite compartirla y sobre todo limitarse a evaluarla una sola vez como máximo; no hay que olvidar que la reducción transformará esta expresión en un valor elemental en la primera ocasión. Gracias a este mismo mecanismo, las expresiones estáticas solo serán evaluadas una sola vez durante toda la existencia del programa.

El algoritmo de reducción usa una pila de evaluación. Este algoritmo consiste a empilar el nodo tratado mientras se trate de una aplicación y a tratar luego el hijo izquierdo; cuando se trata de una hoja, tres casos pueden presentarse:

a) La hoja es una constante, que es retornada directamente.

b) La hoja es un combinador. La regla de reducción correspondiente es aplicada; los parámetros necesarios se encuentran en la pila de evaluación.

c) La hoja es el nombre de una función. La hoja es en este caso sustituída por el cuerpo de la función y la reducción reactivada sobre éste. Las funciones predefinidas son tratadas como combinadores especiales, por razones de eficiencia.

4.2. DESCRIPCION DEL SISTEMA REALIZADO.

Nuestra implementación ha sido realizada en base a un intérprete, que se encarga de traducir las formas KRC en un árbol de combinadores y de reducirlo mediante las reglas presentadas en el párrafo anterior. El intérprete está acompañado por un ambiente de trabajo que permite conservar las informaciones y comunicar con el exterior: definición de funciones, evaluación de expresiones, entradas/salidas, etc.

Tres elementos principales constituyen el intérprete:

a) Análizador, encargado del análisis lexical y sintáctico de las ecuaciones KRC así como de ciertas verificaciones contextuales. La ecuación analizada es traducida en una representación intermediaria, que constituye una versión currificada y listificada de la ecuación ; esta representación es mucho más adaptada a las tareas de abstracción y optimización. La sintaxis usada es de tipo LL(1), lo que nos permitió usar un análisis de tipo descendente recursivo.

b) Abstracción, cuyo objetivo es la supresión de las variables de la ecuación analizada y su transformación en una expresión combinatoria; esta expresión es optimizada a medida que va siendo construída. La abstracción es efectuada al momento de definición de una función o de un conjunto, con respecto a las variables que éstos contienen. Todos los combinadores presentados en el párrafo "Técnica de evaluación" son utilizados por nuestro algoritmo.

c) Reducción, que se encarga de la evaluación de las expresiones simples. La evaluación es efectuada mediante el algoritmo de reducción presentado anteriormente ; cabe indicar que la reducción de un combinador es realmente implementada por una transformación física del árbol que corresponde a la expresión. Los operadores de base del lenguaje son tratados como si fueran combinadores ; es interesante indicar que éstos han sido implementados de manera *inteligente*, tratando de retardar lo máximo posible la evaluación de los elementos de una lista.

El ambiente de trabajo está igualmente compuesto por tres elementos:

a) Nivel superior (*toplevel*). Se trata del elemento de control del intérprete, encargado de leer y de ejecutar las ecuaciones tipeadas por el usuario ; esta función es implementada, a la imagen del *toplevel* LISP, mediante un loop compuesto de tres operaciones: *read* (lectura de una ecuación), *eval* (evaluación de la expresión o definición de la función) y *print* (impresión del resultado). La comunicación con el usuario es efectuada a través del terminal pero puede ser redirigida hacia un archivo.

b) Biblioteca de funciones. Esta biblioteca contiene el conjunto de funciones standard de KRC más algunas funciones propias a nuestra implementación. La mayor parte de estas funciones han sido escritas en Franzlisp por razones de eficiencia. Cabe indicar que todas estas funciones pueden ser redefinidas por el usuario.

c) Comandos. Estos comandos permiten al usuario dos tipos de acción: la comunicación con el exterior (redirigir las entradas, salvar la sesión en un archivo, lanzar un *shell*, etc.) y la consulta o modificación de ciertas variables propias al sistema KRC (nivel y longitud de impresión de una lista, pila de llamadas, etc).

4.3. MEJORAS Y EXTENSIONES EFECTUADAS.

Las experiencias efectuadas nos llevaron a cuestionar ciertos aspectos de KRC y a proponer ciertos otros. La mayor parte de estas mejoras y extensiones fueron agregadas a nuestra implementación:

a) Definición precisa de la noción de conjunto.

b) Construcción de tipo *let*, que permite definir un conjunto de variables locales a una expresión.

c) Funciones anónimas (expresiones lambda), que pueden ser definidas y usadas en una expresión cualquiera.

d) Generalización del mecanismo de filtrado: enriquecimiento de los tipos de filtro existentes y extensión de este mecanismo a la definición de variables de un conjunto o de un *let*.

e) Implementación de las entradas/salidas usando la técnica de flujos (*stream*). Nuestra implementación de esta técnica es bastante simple y tendrá que ser completamente rediseñada en una versión futura.

Otras extensiones no han sido tomadas en cuenta por razones de factibilidad. Entre ellas podemos mencionar: nuevas estructuras de datos, funciones de memorización, funciones de modificación parcial de una estructura de datos, etc.

5. HACIA UN LENGUAJE FUNCIONAL "UTILISABLE".

Los lenguajes funcionales constituyen en nuestra opinión una solución posible a la crisis que sufre actualmente el área de producción de software. Sus bases sólidas, su alto poder de expresión y su paralelismo potencial son tres virtudes inapreciables; KRC constituye una prueba fehaciente de nuestra aseveración. Sin embargo, consideramos que aún queda mucho camino por andar para que esta posibilidad se convierta en realidad; tal vez, la tarea más urgente es la adaptación de estos lenguajes a las necesidades específicas de la producción de software, necesidades que han sido por lo general ignoradas durante el diseño de estos lenguajes.

Un primer imperativo que deberá ser respetado es la eficiencia. Efectivamente, nos parece prácticamente imposible llegar a explotar las aplicaciones tradicionales de la Informática con las performances que ofrecen hoy en día los lenguajes funcionales; este problema es aún más crítico en lo que concierne el software de base y las nuevas aplicaciones informáticas. Por lo tanto, nos parece indispensable mejorar sustancialmente esta eficiencia y, si es posible, llevarla hasta un nivel comparable al de los lenguajes actuales de producción (Cobol, Pascal, C, etc). Las investigaciones realizadas actualmente en este sentido son bastante positivas y nos permiten esperar que este objetivo será alcanzado en un plazo relativamente corto; nuestro trabajo se sitúa en esta línea.

Desde el punto de vista de la programación, un número importante de construcciones y de facilidades hacen falta en los lenguajes funcionales para que aplicados a un contexto de producción. Aquellas que nos parecen más importantes son las siguientes:

a) Mecanismos de abstracción que permitan definir y manipular con mayor facilidad las estructuras de datos: modificarlas parcialmente, compartirlas, etc. Agregar estos mecanismos presenta un problema serio de incompatibilidad entre las propiedades de los lenguajes funcionales y la eficiencia necesaria a la producción de software. Un compromiso es por consiguiente necesario.

b) Nociones de objeto y de módulo. Estas nociones nos parecen indispensables pues sólo ellas permiten definir programas que ofrecen *servicios* a otros programas. Estas nociones están igualmente en conflicto con las propiedades de base de los lenguajes funcionales pues implican la noción de estado.

c) Mecanismos más completos para el tratamiento de las entradas/salidas. Los mecanismos existentes son por lo general bastante pobres, debido a sus características poco funcionales. Limitar estos mecanismos equivaldría a excluir muchos tipos de software del área de aplicación de los lenguajes funcionales.

d) Interface con el mundo exterior. Los sistemas funcionales disponibles hoy en día son excesivamente cerrados sobre ellos mismos; efectivamente, son raros los lenguajes funcionales que permiten usar las rutinas de servicio del sistema operativo y, menos aún, programas escritos en otros lenguajes.

e) Mecanismos de tratamiento de excepciones. Estos mecanismos son bastante útiles en cierto tipo de aplicaciones, en las cuales es necesario tratar directamente los diferentes casos de error, sin la intervención del lenguaje de desarrollo o del sistema operativo.

Otro aspecto que deberá ser estudiado con mucho cuidado es el ciclo de producción que corresponde a este tipo de lenguajes, así como las herramientas necesarias a cada una de las etapas de este ciclo. Este trabajo de exploración y definición deberá, en nuestra opinión, inspirarse de los trabajos efectuados en el área de los ambientes de programación.

Finalmente, consideramos que es también indispensable definir una metodología de programación propia a los lenguajes funcionales. Efectivamente, los programadores acostumbrados a los lenguajes algorítmicos encuentran muchas dificultades cuando programan en un lenguaje funcional; una metodología apropiada les permitiría adaptarse con mayor facilidad y beneficiar al mismo tiempo de una escritura más sistemática de sus programas.

6. CONCLUSION.

Hemos presentado en este artículo los resultados de un proyecto de investigación sobre la adecuación de los lenguajes funcionales a la expresión de problemas informáticos clásicos. Este proyecto ha sido realizado en torno al lenguaje KRC, usado como soporte de experimentación en todas nuestras experiencias ; este lenguaje fue escogido principalmente por su elegancia y sencillez. Cabe indicar que este proyecto es el fruto de una colaboración entre la Escuela de Computación de la Univ. Central de Venezuela y el Laboratorio de Ingeniería Informática de Grenoble.

Dos aspectos esenciales han sido desarrollados en nuestra presentación. El primero corresponde a una evaluación de KRC y, por medio de él, de los lenguajes funcionales; numerosas aplicaciones fueron escritas en este lenguaje para efectuar la evaluación; nuestro artículo presenta únicamente los resultados finales de estas experiencias. El segundo aspecto corresponde a la implementación de KRC realizada para efectos de nuestro proyecto.

Por otro lado, nuestro artículo presenta los puntos que nos parece indispensable desarrollar para que estos lenguajes sean realmente *utilisables* en un contexto de producción de software. La falta de eficiencia es probablemente una de las conclusiones más importantes, lo que nos conduce a pensar que su mejora constituye la tarea más urgente; nuestro trabajo se orienta actualmente en esta dirección. Dos aspectos son estudiados por el momento: la compilación y la evaluación paralela del lenguaje; otro aspecto que podría ser estudiado es una implementación hardware (micro-programada o VLSI) de la técnica de los combinadores.

BIBLIOGRAFIA

[Baker 78] Baker H.G, "Shallow binding in Lisp 1.5", CACM 21-7, Julio 1978, pp. 565-569

[Briat 85] Briat J, Carrier F, Rouzaud Y, Santana M, "A la recherche du langage de programmation de la cinquième génération", Congreso AFCET, Octubre 1985, París

[Carrier 85] Carrier F, Rodriguez E, Santana M, "Agréments et péripéties de la programmation fonctionnelle. Notre expérience avec KRC", T.R. 562, LGI/IMAG, Noviembre 1985, Grenoble

[Clarke 80] Clarke T.J.W, Gladstone P.J.S, MacLean C.D, Norman A.C, "SKIM - The S, K, I Reduction Machine", Proceedingsd of the 1980 Lisp Conference, Agosto 1980, Stanford, pp. 128-135

[Curry 58] Curry H.B, Feys R, "Combinator logic", 1958, North-Holland, Amsterdam

[Friedman 76] Friedman D, Wise D, "CONS should not evaluate its arguments", Automata language and programming, ed. Michaelson and Milner, 1976, Edimburgh Univ. Press, pp. 257-284

[Landin 64] Landin P.J, "The mechanical evaluation of expressions", Computer Journal 6-4, Enero 1964, pp. 308-320

[O'Donnell 85] O'Donnell J.T, "An architecture that efficiently updates associative aggregates in applicative programming languages", IFIP Conference on Functional Languages and Computer Architecture, Septembre 1985, Nancy, pp. 164-189

[Rodriguez 85] Rodriguez E, "Mise en oeuvre de langages fonctionnels. Application á KRC", Memoria de DEA, Universidad de Grenoble, Junio 1985, Grenoble

[Stoye 84] Stoye W.R, Clarke T.J, Norman A.C, "Some practical methods for rapid combinator reduction", ACM Symposium on Lisp and Functionnal Programming, Austin, Agosto 1984, pp. 159-166

[Turner 79] Turner D.A, "A new implementation technique for applicative languages", Software-Practice & Experience 9-9, Setiembre 1979, pp. 31-49

[Turner 81] Turner D.A, "The semantic elegance of applicative languages", ACM Conf. on Functionnal Prog. Languages and Computer Architecture, Portsmouth, Octubre 1981, pp. 85-92

[Turner 82] Turner D.A, "Recursion equations as a programming language", en Functionnal Programming and its Applications, ed. Darlington/Henderson/Turner, Indiana University, 1982

IMPLEMENTAÇÃO DO SISTEMA PASCAL CONCORRENTE NA MAQUINA MICRO-BIS

Simão Sirineo Toscani, Thadeu Botteri Corso, Celso Maciel da Costa
Universidade Federal do Rio Grande do Sul
Porto Alegre - *Brasil*

1. INTRODUÇÃO

Este artigo descreve um sistema (hardware e software) adequado para a construção e execução de programas concorrentes. Trata-se de uma máquina denominada MICRO-BIS, com dois processadores e três módulos de memória, que é adequada para a execução de programas escritos em Pascal Concorrente. O sistema aqui descrito foi concebido a partir das sugestões apresentadas por Brinch Hansen /HAN 78/ e está sendo construído no Curso de Pós-Graduação em Ciência da Computação da UFRGS. No restante desta seção serão apresentados conceitos básicos que facilitam o entendimento do artigo.

Em muitas aplicações de computadores, o sistema (hardware e software) deve ter a capacidade de executar uma variedade de tarefas que são requisitadas de forma imprevisível. Programas concorrentes permitem organizar sistemas deste tipo, convenientemente, de modo a habilitá-los ao atendimento "simultâneo" de várias requisições de serviço, conforme será explicado brevemente a seguir.

Um programa concorrente é constituído por vários módulos (processos seqüenciais) que, por serem relativamente independentes, podem ser executados em paralelo. A execução paralela desses módulos pode ser real (física), quando cada processo é executado por um processador próprio, ou virtual (lógica), quando os processos são executados de forma multiplexada por um único processador. Assim, para permitir o atendimento simultâneo de várias requisições de serviço, basta fazer com que cada tarefa seja executada por um dos processos do sistema.

Os sistemas operacionais de computadores são exemplos de sistemas que possibilitam, aos usuários, a execução de diversos tipos de tarefas (edição de arquivos, compilação e execução de programas, por exemplo), as quais podem ser requisitadas de forma imprevisível. De fato pode-se generalizar afirmando que os sistemas operacionais tem por função o atendimento de múltiplos usuários que requisitam a execução de seqüências de tarefas de forma imprevisível. A execução simultânea de múltiplas tarefas permite que os recursos computacionais (hardware e software) sejam utilizados de forma mais econômica e eficiente. Hoje em dia, mesmo em máquinas de pequeno porte, a maioria dos sistemas operacionais já permite a execução simultânea de várias tarefas.

Muitas aplicações importantes de computadores apresentam a mesma característica de ter que atender a uma variedade de requisições de serviço que ocorrem de forma imprevisível (sistemas para controle "on-line" de informações,

como contas bancárias e estoques, sistemas para reservas de passagens, sistemas para controle de processos industriais, etc). Todas essas aplicações podem ser mais eficientemente implementadas com o auxílio de linguagens de programação concorrente.

As linguagens de programação concorrente também estão estimulando a construção de sistemas operacionais orientados para aplicações específicas, os quais, evidentemente, são mais simples, mais econômicos, mais eficientes e mais confiáveis que os sistemas de propósitos gerais (oferecidos pelos fabricantes) capazes de executar uma grande variedade de serviços que nem sempre são utilizados.

Em um programa concorrente, os processos são relativamente autônomos, mas não independentes. Eles necessitam interagir para trocar informações ou competir por recursos. As linguagens de programação concorrente devem, portanto, prover mecanismos que facilitem a programação das interações entre os processos. Pelo fato dos monitores serem os mecanismos de interação utilizados na linguagem Pascal Concorrente, para a qual é orientado o sistema descrito neste artigo, os mesmos serão apresentados, informalmente, a seguir.

O conceito de monitor foi utilizado pela primeira vez por Brinch Hansen /HAN 70/ no projeto de um sistema operacional multiprogramado. Desde que este conceito foi formalizado por Hoare /HOA 74/, o mesmo vem adquirindo importância cada vez maior, já tendo sido incluído, em várias linguagens de programação concorrente (CSP/K, Módula, Pascal Concorrente e Euclid Concorrente, por exemplo). Os monitores são mecanismos de alto nível para sincronização e intercomunicação de processos, de fácil utilização, que impõem uma estruturação aos programas concorrentes que os utilizam. Um monitor é, na realidade, uma estrutura de dados abstrata com facilidades para sincronização de processos. Pode ser visto como um bloco que contém internamente dados locais e procedimentos para manipular esses dados.

Os dados declarados dentro de um monitor só são acessíveis nos procedimentos locais ao monitor, os quais são executados de forma mutuamente exclusiva, quando chamados pelos processos. Um monitor garante, portanto, exclusão mútua na manipulação desses dados. Além dos tipos de dados usuais, um novo tipo pode ser usado: CONDITION. Variáveis desse tipo só podem ser declaradas dentro de monitores e só podem ser utilizadas através de duas operações especiais: WAIT e SIGNAL. Estas operações permitem que processos sejam colocados à espera de condições que serão sinalizadas por outros processos.

2. O SISTEMA PASCAL CONCORRENTE

O Pascal Concorrente é uma linguagem de programação de alto nível, criada com o objetivo de permitir a cons

trução de sistemas operacionais e programas de controle em tempo real, confiáveis, para computadores de pequeno e médio porte, com um mínimo de esforço e tempo. A linguagem permite a programação de processos concorrentes capazes de se comunicar através de monitores. Segundo Brinch Hansen /HAN 77/, o uso do Sistema Pascal Concorrente é capaz de reduzir em uma ordem de grandeza o esforço despendido na implementação de um sistema concorrente, justificando, assim, a política de confecção de sistemas operacionais dedicados, orientados para atendimento às peculiaridades de aplicações específicas.

O Sistema Pascal Concorrente é composto pelos seguintes elementos:

- Um compilador para a linguagem Pascal Seqüencial SPASCAL /HAR 77/;
- Um compilador para a linguagem Pascal Concorrente, denominado CPASCAL /HAR 77/;
- Um interpretador para o código de uma máquina virtual /MED 81/;
- Um núcleo de sistema operacional /MED 81/.

Os compiladores do sistema, encontram-se disponíveis no código da máquina virtual. Isto significa que ao implementar-se o interpretador e o núcleo da máquina virtual, automaticamente se obtém o ambiente de execução necessário para se dispor dos compiladores Pascal Seqüencial e Concorrente e de todos os programas escritos nessas linguagens. Em particular, o sistema operacional SONIX /COS 85/, compatível com o UNIX, escrito em Pascal Concorrente, poderá ser utilizado, sem alteração alguma, como sistema operacional da máquina MICRO-BIS.

2.1 A Linguagem SPASCAL

A linguagem SPASCAL é baseada na definição de Wirth /JEN 75/, com pequenas restrições e variações. É uma linguagem procedural, simples e estruturada, com construções que permitem a definição de tipos de dados, manipulação de cadeias de caracteres, operações entre registros de mesmo tipo, criação dinâmica de variáveis em uma área de trabalho denominada heap, chamada de procedimentos recursivos, além das operações aritméticas e de controle usuais em linguagem de alto nível.

SPASCAL não possui funções intrínsecas da entrada e saída de dados. Estas funções devem ser implementadas pelo sistema operacional e comunicadas ao compilador SPASCAL através de uma porção do programa fonte denominado prefixo. Essa característica facilita o desenvolvimento de novos sistemas operacionais com funções de entrada e saída diferentes, pois os compiladores não estão, à princípio, vinculados com nenhum sistema operacional específico.

2.2 A Linguagem CPASCAL

A linguagem CPASCAL foi projetada especialmente para permtir a escrita de programas concorrentes que cumprem as funções básicas de sistemas operacionais, Foram feitas restrições e extensões a SPASCAL, chegando-se a um modelo de definição de processos e monites semelhante ao formalizado inicialmente por Hoare /HOA 74/.

Um programa concorrente é geralmente composto por um conjunto de processos que compartilham o processador, sob a gerência do núcleo o sistema operacional. Tais processos podem comunicar-se através de monitores, que representam os mecanismos que garantem o acesso exclusivo às seções críticas do programa concorrente /HOL 78/. Os processos possuem também construções que permitem a passagem de controle de execução para programas seqüenciais, escritos em SPASCAL, e carregados em áreas específicas dos processos. Uma vez em execução, os programas seqüenciais podem solicitar serviços aos processos que os supervisionam, através das interfaces contidas no prefixo.

A linguagem CPASCAL não permite procedimentos recursivos nem criação dinâmica de variáveis, para evitar crescimento incontrolado dos dados de execução. O conceito de classe introduz a utilização de reentrância de código objeto para monitores e processos concorrentes. A especificação completa da linguagem CPASCAL está descrita em /HAN 77/.

2.3 O Interpretador e o Núcleo da Máquina Pascal Concorrente

Os compiladores produzem código objeto para uma máquina virtual (Máquina Pascal Concorrente - MPC), com 112 isntruções. A palavra da MPC é de 16 bits e cada código de operação ou operando de instrução ocupa uma palavra de memória. O esquema de endereçamento permite acesso a caracteres de 8 bits, sendo, portanto, de 64 Kbytes o espaço de endereçamento virtual.

O núcleo contém funções primitivas, para compartilhamento de tempo de processamento, contagem do tempo real, exclusão mútua no acesso a monitores, comunicação entre processos e operações de entrada e saída.

Para execução de operações de entrada e saída no sistema existe uma única operação, denominada IO, com operandos que especificam o tipo de operação, o dispositivo de entrada e saída envolvido e os dados a serem transferidos pela operação. Uma instrução IO é executada de forma síncrona, isto é, a instrução que a segue no programa objeto só é executada após toda a operação de entrada e saída ter sido completada. Esta característica influiu fortemente na escolha da arquitetura para o MICRO-BIS, especialmente nos aspectos relacionados a entrada e saída dos dados.

O Sistema Pascal Concorrente, conforme definido por Brinch Hansen, pressupõe implementações em instalações com um único processador. Obviamente, para o caso do MICRO-BIS, onde existirão dois processadores, o núcleo deverá ser diferente.

O núcleo implementa também memórias virtuais para os processos. A memória virtual de cada processo é constituída por um segmento comum (igual para todos os processos) e um segmento privativo. O segmento comum contém o interpretador, o núcleo e o código virtual do programa concorrente em execução. O segmento privativo de um processo (segmento de dados) contém o stack e o heap desse processo.

3. ARQUITETURA DA MÁQUINA BIMICROPROCESSADORA

A máquina bimicroprocessadora MICRO-BIS, mostrada na figura 1, é formada por dois microprocessadores MC68000, uma memória local de 128 Kbytes para cada microprocessador e uma memória global, também de 128 Kbytes. Esses componentes estão dispostos em uma placa que deve ser ligada a um minicomputador ED-281 com a seguinte configuração: CPU Z8ØA, 2 terminais, 112 Kbytes de memória RAM, 2 unidades de discos flexíveis de 8", 1 unidade de disco rígido tipo winchester e 1 impressora de 1ØØ caracteres por segundo. O minicomputador ED-281 será o processador de entrada e saída (PES) do sistema.

Cada microprocessador MC68000 pode acessar a sua memória local e a memória global. Dessa forma a intercomunica e troca de mensagens entre processos rodando em processadores diferentes somente pode ser realizada através da memória global.

Para o ED-281 poder realizar a sua função no sistema (execução das operações de entrada e saída) o mesmo deverá poder acessar a memória particular de cada microprocesador MC68000. Cada microprocessador MC68000 poderá interromper o outro e o ED-281, sendo que o último poderá interromper ambos os microprocessadores MC68000.

4. DISTRIBUIÇÃO DOS COMPONENTES DA MPC NA MÁQUINA REAL

A idéia que norteou a definição da divisão dos componentes da MPC na máquina real baseou-se no fato de que cada processador deve operar em sua memória privativa o máximo de tempo possível, com isso minimizando os acessos à memória global, o que conduz a um melhor desempenho do sistema.

Para tanto decidiu-se manter em cada memória privativa dos processadores uma cópia do segmento comum aos processos (núcleo, interpretador e o código virtual gerado pelo compilador Pascal Concorrente) e na memória global o segmento de dados do processo inicial, os dados locais dos

monitores e as estruturas de dados que implementam os monitores (um monitor é implementado por uma variável booleana que indica se o mesmo está livre ou não e uma fila onde ficam os processos a espera para entrar no monitor).

Na memória local de um processador também devem ser alocados os segmentos privativos dos processos que vão ser executados por esse processador. Como em Pascal Concorrente a intercomunicação de processos é feita através de monitores, o acesso à memória global somente é necessário quando um processo deseja acessar um monitor.

Com a duplicação do segmento comum nas memórias locais existe uma perda em relação a memória que poderia ser destinada aos segmentos privativos dos processos. Porém, se for considerado que cada memória local possui 128 Kbytes, que o núcleo e o interpretador somam juntos aproximadamente, 12 Kbytes e, que o código virtual do sistema operacional SONIX (que será o sistema utilizado na máquina) tem 18 Kbytes, resta em cada memória local 98 Kbytes. Supondo que o sistema operacional seja definido com 10 processos (a presente implementação da MPC suporta até 16), sendo alocados 5 para cada processador, então cada segmento privativo poderia ter aproximadamente 20 Kbytes.

5. SISTEMA DE ENTRADA E SAÍDA

A disparidade de velocidade entre dispositivos de entrada e saída e processador obriga a utilização de processadores de entrada e saída, isto é, dispositivos que visam liberar a CPU de executar tais operações.

Conforme já foi mencionado anteriormente, o sistema de entrada e saída será implementado no microcomputador ED-281 (o qual será referido como PES), sob o sistema operacional MP/M(Multi-programming monitor control program).

As rotinas de processamento de entrada e saída são freqüentemente as mais complexas num sistema operacional. Isto é devido, em parte, a natureza assíncrona dos dispositivos de entrada e saída e aos problemas de sincronização causados pelo paralelismo inerente da sua operação. Na solução adotada optou-se por implementar atendimento seqüencial das requisições de entrada e saída geradas pelos processos.

A seguir o ambiente do sistema é descrito caracterizando-se inicialmente as estruturas de dados e posteriormente sua filosofia de funcionamento.

Conforme a figura 2, em cada memória particular dos processadores MC68000 haverá uma fila de requisições de entrada e saída e uma fila de operações já executadas pelo PES. Os elementos dessas filas são estruturados da seguinte maneira:

```
type REQUISIÇÃO = record
                      NÚMERO-PROCESSO
                      PERIFÉRICO
                      TICKET
                      OPERAÇÃO
                      ENDEREÇO-LEITURA
                      ENDEREÇO-ESCRITA
                      STATUS
                  end;
```

O campo ticket foi utilizado para garantir que o atendimento das requisições de entrada e saída seja realizado na ordem em que as mesmas foram solicitadas pelos processos, seqüencializando desta forma as operações.

A estrutura que armazenará as requisições à medida em que forem geradas, é definida da seguinte maneira:

```
type FILA = array(Ø..15) of REQUISIÇÃO
```

À medida que as requisições forem sendo atendidas elas irão passando para a fila de operações executadas, representada na mesma estrutura acima.

A capacidade do array FILA foi estabelecida em 16 considerando-se que o número máximo de processos do sistema operacional a ser implementado está definido em 16 e que pode haver uma situação em que todos os processos sejam alocados em um único processador (a alocação é determinada em tempo de carga conforme será explicado na seção 6).

Na memória global aos processadores MC68000 haverá uma variável chamada TICKET que, inicializada com Ø, é incrementada pelo núcleo a cada requisição de entrada e saída realizada pelos processos. Com isto pode-se implementar ordem seqüencial de atendimento.

No PES não é necessária a definição de uma fila, pois a ordem de atendimento das requisições de entrada e saída será determinada pelo campo ticket que o processo "toma" (conforme acima citado) ao requisitar uma operação de entrada e saída. Dessa forma, ao ser interrompido, o PES verifica qual será a próxima requisição a ser atendida, obtendo os parâmetros para tratar a operação da fila na memória local do processador que causou a interrupção, garantindo assim uma política justa de atendimento.

A figura 3 mostra as diversas seções em que é dividido o sistema, as quais são a seguir comentadas.

Um processo quando requisita uma operação de entrada e saída perde o processador e é inserido na fila de processos bloqueados até a operação ter sido completada, quando então é colocado fila de processos prontos para novamente concorrer pelo processador. Entretanto, a ação de bloquear

os processos não faz parte do procedimento de requisição de entrada e saída. O interpretador ao reconhecer uma instrução deste tipo, realiza uma chamada ao núcleo do sistema operacional. Este por sua vez coloca os parâmetros na fila de requisição na sua respectiva memória local e interrompe o PES, executando a seguinte instrução:

```
MOVE $2,$4ØØØØ ; carrega no endereço 4ØØØØH o comando de
               ; interrupção do PES
```

Essa instrução faz com que o hardware transfira o valor armazenado na posição 4ØØØH (registrador de comandos do MC68000) para o registrador de status (E4H) do PES, onde a requisição é indicada pelo nível lógico 1 do bit correspondente ao processador (bit Ø corresponde a PØ e bit 1 corresponde a P1). Essa ação ocasiona uma interrução IRQ14 que no PES é controlada por dois elementos PIC 8259 (Programmable Interrupt Controller), ligados na configuração mestre-escravo. A figura 4 mostra a arquitetura do sistema de forma mais detalhada, o que permite um melhor entendimento do funcionamento do mesmo.

A rotina de atendimento de interrupção, ao ser disparada, consulta o registrador de status e liga uma variável de controle indicando assim ao PES a existência de uma requisição a ser atendida.

O PES, através de uma rotina específica, transfere os parâmetros da fila na memória local do processador que o interrompeu para o seu registro de operação (na memória), que apresenta a seguinte estrutura:

```
type REGISTRO-DE-OPERAÇÃO = record
                                NÚMERO-DO-PROCESSO
                                PERIFÉRICO
                                OPERAÇÃO
                                ENDEREÇO-LEITURA
                                ENDEREÇO-ESCRITA
                                PROCESSADOR
                                STATUS
                                VARIÁVEL-CONTROLE
                            end;
```

Caso o PES esteja realizando uma operação de entrada e saída, esta será completada para após ser consultada a variável de controle. Se esta estiver ligada, serão consultadas as filas para verificar qual a próxima requisição a ser tratada. Note que esta variável permanecerá ligada enquanto houver requisições nas filas.

O endereçamento para transferência de dados de/para as memórias locais dos MC68000 é feito utilizando-se dois registradores de endereçamento (EØH e E1H), cujas estruturas são mostradas na figura 5. Como as memórias estão divididas em 8 K páginas de 16 bytes, são necessários 13 bits pa

ra endereçá-las (as páginas). Esses bits são obtidos através da concatenação dos 5 bits menos significativos do registrador EØH com os 8 bits do registrador ElH. O deslocamento dentro da página, ØH a FH, é obtido a partir dos 4 bits menos significativos das portas de endereço FØH a FFH do PES. O bit 5 do registrador EØH permite selecionar a memória local que se deseja referir.

Realizada a operação de entrada e saída, será devolvido uma indicação do resultado da mesma fila de status (ocupando o mesmo nodo em que estava representada a requisição, na estrutura da fila mostrada na figura 2) na memória local do processador que requisitou a operação. Esse processador será avisado do término da operação através de uma interrupção gerada pelo PES via registrador de comandos (E2H). Isto é feito através das seguintes instruções:

```
LD A, XH      ; carrega o acumulador com o comando desejado
OUT (ØE2H), A; escreve no registrador de comandos o comando
              ; armazenado no acumulador
```

onde X pode assumir os valores Ø ou 1 conforme o processador em questão.

Com isto o processador interrompido devolve o status da operação realizada para o processo que a requisitou, colocando o mesmo na fila de processos aptos.

Além da interrupção, o registrador de comandos do PES permite a execução de operações HALT e RESET nos processadores PØ e Pl (figura 6). Isto é feito de maneira semelhante ao procedimento de interrupção, utilizando-se os mesmos comandos acima descritos, mudando contudo o valor de X, conforme o comando desejado. Assim para executar uma operação HALT, X deverá assumir os valores 4H ou 8H, respectivamente, para PØ ou Pl. Da mesma forma, para executar a operação RESET, X assume os valores 1ØH (PØ) ou 20H (Pl).

O driver de entrada e saída é formado pelas rotinas de atendimento das requisições para cada periférico disponível no sistema, sendo sua estrutura básica definida da seguinte maneira:

```
CARGA-DO-SISTEMA;

loop
  if VARIÁVEL-CONTROLE
    then begin
         CARREGA-REGISTRO-DE-OPERAÇÃO;
         case REGISTRO-DE-OPERAÇÃO.PERIFÉRICO of
             "terminal"    : ...
             "disquete"    : ...
             "winchester"  : ...
             "impressora"  : ...
           else            : nada;
         end case;
         end
forever;
```

O sistema operacional é carregado nas memórias locais dos processadores através da rotina CARGA-DO-SISTEMA. Feito isto, é enviado um sinal de reset para ambos os processadores, fazendo com que estes comecem a executar. Realizado o procedimento inicial de carga, o PES entra em estado de busy-wait até ser interrompido por algum dos processadores, para atender uma requisição de entrada e saída.

6. INICIALIZAÇÃO DO SISTEMA

A inicialização do sistema é realizada a partir do PES. Incialmente deve ser executado o programa que implementa o sistema de entrada e saída. Este programa dispara o procedimento inicial de carga cuja função é transferir de disco para as memórias locais, em posições físicas pré-definidas, os códigos que implementam a MPC e o programa concorrente. Ao término da transferência da MPC e do sistema operacional para as memórias locais, cada processador recebe um comando reset que o habilita a entrar em estado de processamento.

Ao iniciar a execução, o processador acessa uma posição de memória que contém o endereço de um procedimento que estabelece o protocolo de conversação com o usuário, através do qual será definido quais os processos que deverão ser executados pelo processador. A identificação do processo deve ser feita pelo número que indica a ordem de execução da instrução INIT para esse processo no programa concorrente.

Ao término da associação de processos a processadores, o sistema estará apto a operar normalmente o interpretador passará a executar as instruções virtuais do programa concorrente.

7. CONCLUSÕES

O trabalho descrito neste artigo faz parte de um projeto em andamento. A arquitetura apresentada já tem seu projeto lógico concluído. Atualmente está sendo feita a fixação dos componentes na placa. O tempo estimado para a conclusão desse trabalho (ligação da placa no computador ED-281 e testes da mesma) é de 4 meses.

O sistema de entrada e saída está todo definido, estando em fase final de programação. As alterações que devem sofrer a MPC de modo a funcionar com 2 processadores já foram todas definidas, estando a mesma em fase de programação. O tempo previso para término da implementação do software é de seis meses, após a conclusão do hardware.

O desenvolvimento do software está sendo levado em paralelo com a construção do hardware. Isto é possível porque se dispõe de um computador ED680 cujo processador principal é um MC68000. Foi escolhida a linguagem C para a programação do sistema, em função da sua disponibilidade no

ED680 e da sua adequação para a produção de software básico.

É importante salientar que o sistema em construção é um protótipo. Todas as soluções adotadas levaram em conta principalmente a simplicidade de implementação para que o tempo de produção do sistema fosse mínimo. Pretende-se com isso, no menor prazo possível, ter o sistema operando, o que servirá de estímulo e incentivo para elaborar um projeto mais ambicioso de construção de uma máquina multiprocessadora.

BIBLIOGRAFIA

/COS 85/ COSTA, C.M. Um sistema operacional compatível com o UNIX para a Máquina Pascal Concorrente. In: XVIII CNI, São Paulo, setembro 1985. Anais. São Paulo, SUCESU, 1985. p.728-732.

/HAN 70/ HANSEN, P.B. The nucleus of a multiprogramming system. Comm. ACM 13(4):238-50, Apr. 1970.

/HAN 77/ HANSEN, P.B. The architecture of concurrent programs. Englewood Cliffs, N.J. Prentice-Hall, 1977.

/HAN 78/ HANSEN, P.B. Multiprocessador architectures for concurrent programs. In: ACM 78, Washington, D.C., Dec. 1978. Proceedings. New York, ACM, 1978. p.317-323.

/HAR 77/ HARTMAN, A.C. A concurrent pascal compiler for minicomputers. Berlin, Springer-Verlag, 1977. (Lectures Notes in Computer Science 50).

/HOA 74/ HOARE, C.A.P. Monitors: an operating systems systems structuring concept. ACM, 17(10):549--557, Oct. 1974.

/HOL 78/ HOLT, R.C. et alii. Structured concurrent programming with operating systems applications. Reading, Addison-Wesley, 1978.

/JEN 75/ JENSEN, K. & WIRTH, N. PASCAL: user manual and report. New York, Springer-Verlag, 1975.

/MED 81/ MEDEIROS, G.C.R. Implementação do sistema PASCAL CONCORRENTE no Labo 8034. Porto Alegre, PGCC da UFRGS, 1981. (Dissertação de Mestrado).

/THO 78/ THOMPSON, K. Unix implementation. The Bell System Technical Journal, 57(6):1931-46, July--August, 1978.

/WEB 82/ WEBER, T.S. & ROCHA, A.C. Descrição da arquitetura da máquina Pascal Concorrente. Porto Alegre, CPGCC-UFRGS, 1982. (relatório interno).

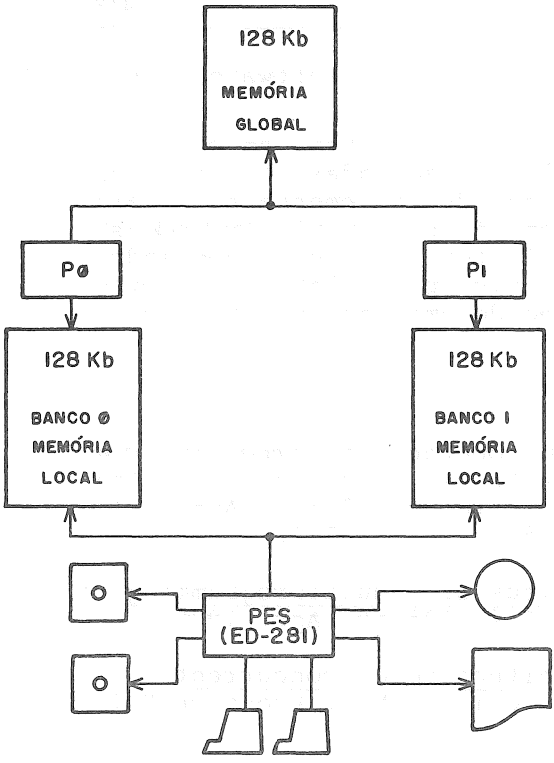


Figura 1. - Arquitetura do Sistema.

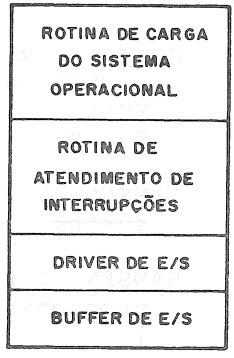
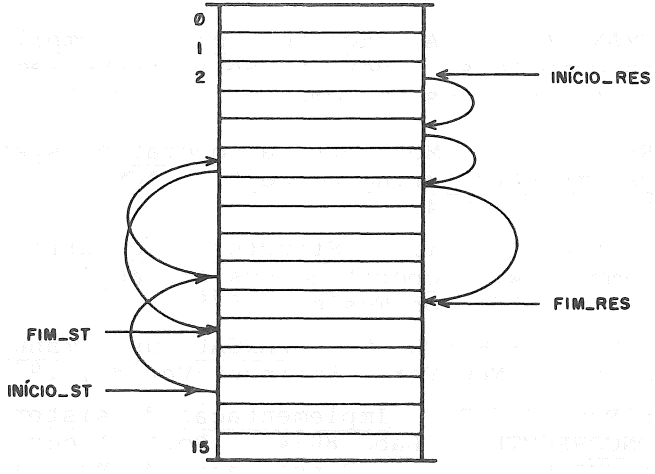


Figura 3. - Estrutura Lógica do Sistema de Entrada e Saída.



INÍCIO-RES: APONTADOR PARA INÍCIO DA FILA DE REQUISIÇÕES DE ENTRADA E SAÍDA;
FIM-RES: APONTADOR PARA FIM DA FILA DE REQUISIÇÕES DE ENTRADA E SAÍDA;
INÍCIO-ST: APONTADOR PARA INÍCIO DA FILA DE STATUS;
FIM-ST: APONTADOR PARA FIM DA FILA DE STATUS.

Figura 2. - Filas de Requisição de E/S e de Status nas Memórias Locais nos Processadores 68000.

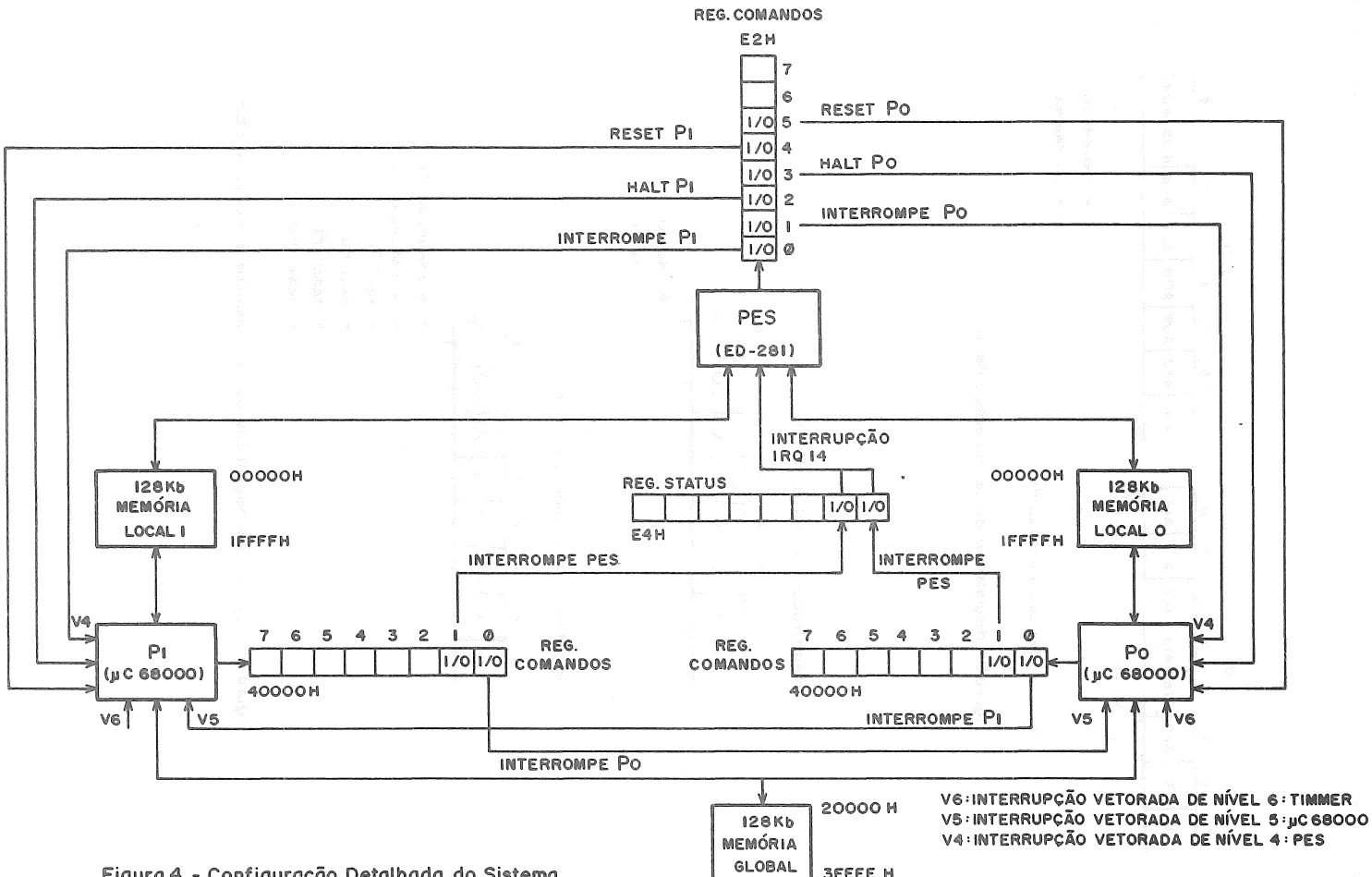


Figura 4. - Configuração Detalhada do Sistema.

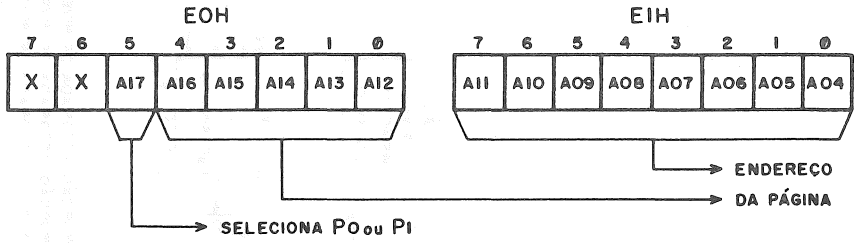
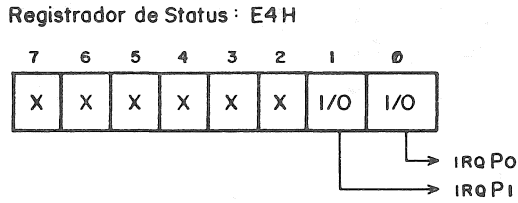


Figura 5- Registradores de Endereçamento.



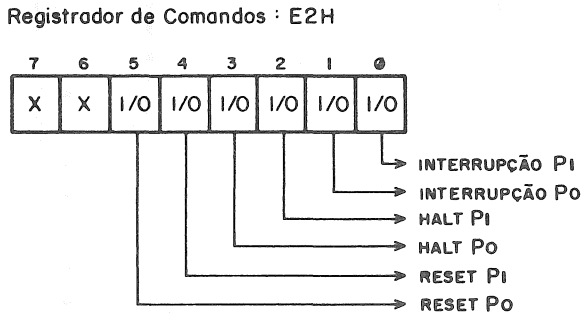


Figura 6- Lay-out dos Registradores de Comandos e Status do PES.

UMA IMPLEMENTAÇÃO DE META ASSEMBLER EM PASCAL

Flávio Soibelmann Glock
Porto Alegre - *Brasil*

- RESUMO

O Meta Assembler é um software que possibilita gerar montadores assembly para diferentes processadores.

A partir da definição de um formato generalizado obtido através do estudo das linguagens dos principais processadores o autor desenvolveu a Linguagem de Definição de Assembler (LDA). Com ela o usuário define o código de máquina a ser gerado e a sintaxe da linguagem que será utilizada.

Aplicando o formalismo LDA é feita a descrição da linguagem assembly do processador Z80.

- ABSTRACT

The Meta Assembler is an utility software which generates assemblers for any microprocessor. The user writes the specifications which define the machine-code to be generated and the language sintax to be used.

The author designed a generalized format to describe the most popular microprocessor languages. It is the Assembly Language Definition (LDA) format.

The LDA language implementation of Zilog's Z80 microprocessor assembly language is shown.

1. INTRODUÇÃO

O estado atual da informática exige que os computadores sejam capazes de "se comunicarem" com o programador em linguagens dirigidas a facilitar o trabalho humano. No entanto, os computadores desenvolvidos para funções específicas não são acessíveis ao programador, uma vez que normalmente não estão conectados a teclados e vídeos. Além disso, quando o programador fornece instruções, estas só podem ser executadas após a tradução para a linguagem de máquina dos processadores usados.

A construção de novos computadores impõe então recursos especiais para desenvolver o software necessário. O META ASSEMBLER é um desses recursos.

2. OBJETIVO

Para desenvolver equipamentos baseados em novos microprocessadores são necessárias ferramentas de software adequadas, as quais nem sem-

pre são disponíveis. O desenvolvimento dessas ferramentas leva à sobrecarga de trabalho e aumenta os prazos e custos dos projetos.

A solução encontrada para minimizar essas dificuldades foi a criação de um software genérico. O objetivo desse software é gerar montadores dedicados para cada microprocessador, a partir das especificações fornecidas.

O estudo das linguagens assembly dos microprocessadores existentes no mercado permitiu a identificação dos seus elementos comuns, fornecendo o princípio geral deste meta assembler.

Entende-se por meta assembler o software capaz de, com as especificações fornecidas, gerar montadores assembly. Estes, identificando uma linguagem, geram código de máquina.

Este software foi desenvolvido com o objetivo de estimular a utilização de microprocessadores novos em projetos de instrumentação no Centro de Engenharia Biomédica da Pontifícia Universidade Católica do Rio Grande do Sul (CEB-PUCRS), bem como fornecer suporte a outros setores de desenvolvimento dentro e fora da Universidade.

3. ASPECTOS GERAIS DO SOFTWARE

3.1. VISÃO GERAL DE UM ASSEMBLER

A expressão "assembler" traduzida literalmente significa "montador". É assim chamada porque se refere a um programa que traduz uma linguagem simbólica simples com a qual representamos operações (ex: LD A,B quando queremos transferir uma informação do registrador B para o registrador A) em linguagem de máquina (ex: 01111000 que representa a operação de transferência efetiva do registrador B para A). O assembler deve compor ou "montar" um código de bits, a partir de tabelas específicas, inerentes a cada processador.

Cada instrução em linguagem simbólica corresponde a um número determinado de bytes em linguagem de máquina. O assembler mantém um contador para registrar a posição onde ficarão esses bytes na memória do processador. Esta posição é chamada de endereço de memória. O assembler permite que o usuário guarde esses endereços colocando nomes ou "etiquetas" específicas para cada um. Esta forma de registro constitui os LABELS.

Além de permitir a definição de labels, o assembler permite a definição do próprio contador de memória, indicando assim onde colocar o programa. Podem ser definidos espaços que serão ocupados por variáveis e também valores iniciais para locais de memória.

O assembler deve ser capaz de interpretar expressões aritméticas e lógicas. Muitas vezes o usuário precisa definir certos operandos a partir de dados que serão estabelecidos durante a montagem do programa. Utiliza-se nesse caso, além das quatro operações, as operações com bits

(shift, separação de bytes) e operações lógicas (e, ou, não, ou-exclusivo). Além disso, o assembler deve aceitar dados em vários sistemas numéricos tais como binário, octal, hexadecimal e caracteres ASCII.

3.2. IMPLEMENTAÇÃO DO META ASSEMBLER NO COMPUTADOR CEB-286-PUCRS

O meta assembler foi escrito em linguagem Pascal. O código objeto foi gerado através de um compilador TURBO Pascal implementado em sistema operacional MS/DOS 3.0.

O Software é constituído de 4 partes:

- módulo de interpretação de expressões numéricas;
- módulo de formatação de saída e de mensagens de erro;
- módulo de interpretação da Linguagem de Definição de Assembler;
- módulo principal. Este módulo implementa um assembler em duas etapas (2-pass assembler).

O código de máquina é gerado em formato hexadecimal, compatível com o programador de EPROM que está sendo desenvolvido no CEB-PUCRS.

3.3. FUNCIONAMENTO DO ASSEMBLER EM DUAS ETAPAS

Para "aprender" a linguagem a ser utilizada o software lê as especificações que estão armazenadas em disco.

O programa a ser montado é lido uma primeira vez. Durante esta primeira passagem o software cria uma lista de labels usados (tabela de labels).

O software lê o programa pela segunda vez, executando as seguintes tarefas:

- indica eventuais erros;
- grava no disco o programa em linguagem de máquina, em formato hexadecimal;
- lista o programa na tela ou impressora.

3.4. UTILIZAÇÃO DO META ASSEMBLER

A especificação da linguagem para o processador é feita através de um editor de texto, de acordo com as regras da Linguagem de Definição de Assembler (LDA).

O programa a ser montado é escrito utilizando-se o editor de de texto ou então é transmitido de outro computador.

Inicia-se a operação do meta assembler indicando as especificações a serem utilizadas e o nome do programa que se deseja montar.

O operador deve optar por saída na impressora/disco ou tela/dis-

co para obter a listagem.

O código de máquina gerado será transmitido para o equipamento objeto da montagem através de programador de EPROM ou via interface serial ou paralela.

3.5. VANTAGENS E LIMITAÇÕES DO SOFTWARE

O META ASSEMBLER permite a montagem de programas para processadores de 8, 16 e 32 bits, sendo que o limite de precisão para os operandos numéricos é de 32 bits mais sinal. O tempo de implementação de um montador para um novo processador é estimado em 40 horas.

O programa é compatível com diversos computadores, uma vez que é escrito em linguagem Pascal.

O tamanho máximo do programa a ser montado depende da capacidade da memória principal, que guarda a tabela de labels mais a definição do processador.

A velocidade de montagem varia dependendo do processador definido, mas situa-se em torno de 10 linhas por segundo utilizando-se o computador CEB-286-PUCRS.

4. A LINGUAGEM DE DEFINIÇÃO DE ASSEMBLER

A linguagem de definição de assembler serve para a codificação das especificações em formato capaz de ser interpretado pelo meta assembler. Para escrever em LDA o programador utiliza um editor de texto, criando um arquivo do tipo PROC.DEF no disco, sendo PROC o nome do processador a ser definido.

Uma linguagem fica definida pela criação de 5 conjuntos de informações:

- tabelas de constantes;
- tabelas de opcodes;
- limites para operandos numéricos;
- nome dos pseudo-opcodes utilizados;
- tabelas de sintaxe para as operações.

4.1. TABELAS DE CONSTANTES

Os operandos do mesmo tipo, como condições e sets de registradores, são utilizados várias vezes com o mesmo opcode, modificando o código de máquina gerado de acordo com tabelas definidas. Estes operandos são colocados para a LDA como tabelas de constantes.

4.2. TABELAS DE OPCODES

As operações que tem a mesma sintaxe, como as operações aritméticas e lógicas, são agrupadas através de tabelas de opcodes. Nestas ta-

belas consta o nome de cada operação e o valor numérico que o mesmo assume na montagem.

4.3. LIMITES PARA OPERANDOS NUMÉRICOS

A verificação da sintaxe em determinadas operações requer o teste de limites das expressões utilizadas a fim de evitar o overflow na montagem. Por exemplo, uma instrução que utiliza um byte de dados não deve ser montada se o valor dado for maior que 255. O usuário define os valores mínimos e máximos para as expressões.

Também são definidos o tamanho da palavra e o tamanho do endereçamento do processador. A definição do tamanho da palavra é importante pois alguns processadores possuem contador de endereços que incrementa 2 bytes de cada vez, enquanto que outros incrementam apenas 1 byte. O tamanho do endereçamento é utilizado pelo módulo de formatação de saída, colocando 2 ou 4 bytes de endereço antes de cada linha de código.

4.4. NOME DOS PSEUDO OPCODES UTILIZADOS

O meta assembler possui diversos opcodes definidos. No entanto, cada fabricante define nomes para as operações executadas pelo assembler de seu processador, tornando necessária a flexibilidade do sistema neste ponto. Por isso a LDA permite a redefinição dos nomes dos opcodes.

4.5. TABELAS DE SINTAXE PARA AS OPERAÇÕES

Através da LDA o usuário define as operações que o assembler deve ser capaz de montar. A definição é feita a partir dos nomes das tabelas já definidas e utilizando símbolos, tais como vírgulas, parênteses e outros, que descreverão a sintaxe da operação.

5. EXEMPLO DE APLICAÇÃO: DEFINIÇÃO DO PROCESSADOR Z80

A equipe do Centro de Engenharia Biomédica definiu o assembler do processador Z80 em LDA, para possibilitar o prosseguimento de um trabalho já iniciado em um computador CP/M onde era utilizado o montador MACRO-80. A vantagem da implementação em LDA é o aproveitamento do hardware existente no computador CEB-286, que possui maior capacidade de armazenamento em disco e onde está sendo desenvolvido um programador de EPROM.

O programa a seguir implementa o assembler Z80.
Uma descrição da linguagem LDA encontra-se no apêndice.

```
; --------------------------------
; definição do processador Z80
; --------------------------------

; definição do pseudo opcodes.

pseudo ORG =        ORG
```

```
pseudo EQU =        EQU
pseudo END =        END
pseudo PHASE =      .PHASE      ; para compatibilidade com Macro-80
pseudo DEPHASE =    .DEPHASE
pseudo DIW =        DEFW
pseudo DEFS1 =      DEFS
pseudo DB1 =        DEFB
pseudo SET =        SET
pseudo TITLE =      TITLE

; definição de tamanho de endereçamento.

word = 1      ;(contador de endereços incrementa para cada 1 bytes)
address = 2   ;(2 bytes de endereçamento)

; definição de intervalos numéricos

range d:    - 127 .. 127
range adrs: 0 .. 0FFFFH
range n:    0 .. 0FFH
range nn:   0 .. 0FFFFH
range p:    0 .. 38H

; definição de tabelas de constantes

qq:       ;a tabela 'qq' e utilizada na montagem
 BC=00H   ; das operações 'PUSH' e 'POP'
 DE=10H
 HL=20H
 AF=30H

rega:
 B=0
 C=8
 D=10H
 E=18H
 H=20H
 L=28H
 A=38H

regb:
 B=0
 C=1
 D=2
 E=3
 H=4
 L=5
 A=7

pp:
 BC=00H
 DE=10H
```

```
 IX=20H
 SP=30H


rr:
 BC=00H
 DE=10H
 IY=20H
 SP=30H

rbd:
 BC=00H
 DE=10H

dd:
 BC=00H
 DE=10H
 HL=20H
 SP=30H

cc:
 NZ=00H
 Z=08H
 NC=10H
 C=18H
 PO=20H
 PE=28H
 P=30H
 M=38H

ix:
 IX=0DDH
 IY=0FDH

cjr:
 C = 38H
 NC = 30H
 Z = 28H
 NZ = 20H

bit:
 0=0H
 1=8H
 2=10H
 3=18H
 4=20H
 5=28H
 6=30H
 7=38H

; definição de tabelas de opcodes
```

```
opcode calc:
 INC=04H
 DEC=05H

opcode arita:
 AND=20H
 CP=38H
 OR=30H
 XOR=28H

opcode aritb:
 ADD=00H
 ADC=08H
 SUB=10H
 SBC=18H

opcode seta:
 BIT=40H
 RES=80H
 SET=0C0H

opcode blt:
 NEG =  $44
 LDI =  $A0
 LDIR = $B0
 LDD =  $AB
 LDDR = $BB
 CPI =  $A1
 CPIR = $B1
 CPD =  $A9
 CPDR = $B9
 RLD =  $6F
 RRD =  $67
 RETI = $4D
 RETN = $45
 INI =  $A2
 INIR = $B2
 IND =  $AA
 INDR = $BA
 OUTI = $A3
 OTIR = $B3
 OUTD = $AB
 OTDR = $CC

opcode rot:
 RLC=0H
 RRC=8H
 RL=10H
 RR=18H
 SLA=20H
 SRA=28H
 SRL=38H
```

```
; definição da tabela de sintaxe e de montagem:
;   a primeira linha define a sintaxe da operação;
;   as linhas seguintes dão instruções para montagem.

JP (HL)    ;representa um instrução com sintaxe rígida
 0E9H      ;monta 1 byte = E9 hexa.

             ;caso a sintaxe 'JP' acima não sirva, então continua
             ;  procurando outra forma da instrução:

JP (ix)    ;representa uma instrução com opções: usa a tabela 'ix'
  ix       ;monta 1 byte = DD ou FD hexa,
 0E9H      ;  e o segundo byte = E9 hexa.

JP cc,nn
 0C2H + cc
 LOW nn
 HIGH nn

JP nn
 0C3H
 LOW nn
 HIGH nn

arita (HL)    ;usa a tabela de opcodes 'arita'
 arita+86H    ;soma o valor encontrado com 86 hexa.
arita regb       ;a ausência de tabulador marca início de definição
 arita+regb+80H ;a tabela de montagem tem sempre tabulador antes.
arita (ix d)
 ix
 arita+86H
 d
arita n
 0C6H+arita
 n

aritb A,(HL)
 aritb+86H
aritb A,regb
 aritb+regb+B0H
aritb A,(ix d)
 ix
 aritb+86H
 d
aritb A,n
 0C6H+aritb
 n
IN rega,(C)
 0EDH
 40H+rega
IN A,(n)
```

```
 0DBH
 n
OUT (C),rega
 0EDH
 41H+rega
OUT (n),A
 0D3H
 n
; ********* LD simples ************
LD A,I
 0EDH
 57H
LD A,R
 0EDH
 5FH
LD I,A
 0EDH
 47H
LD R,A
 0EDH
 4FH
LD SP,HL
 OF9H
LD SP,ix
 ix
 OF9H
LD rega (HL)
 46H + rega
LD rega,regb
 40H+rega+regb
LD (HL),regb
 70H+regb
LD A,(rbd)
 rbd+0AH
LD (rbd),A
 rbd + 2
; ********* LD com o primeiro operando simples **************
LD HL,(nn)
 2AH
 LOW nn
 HIGH nn
LD dd,(nn)
 0EDH
 5BH+dd
 LOW nn
 HIGH nn
LD dd,nn
 dd + 1
 LOW nn
 HIGH nn
LD rega,(ix d)
 ix
```

```
  46H+rega
  d
 LD A,(nn)
  3AH
  LOW nn
  HIGH nn
 LD rega,n
  rega + 6
  n
 LD (HL),n
  36H
  n
 LD ix,(nn)
  ix
  2AH
  LOW nn
  HIGH nn
 LD ix,nn
  ix
  21H
  LOW nn
  HIGH nn
 ; *********** algum cálculo no primeiro operando ***********
 LD (ix d),regb
  ix
  70H+regb
  d
 LD (ix d),n
  ix
  36H
  d
  n
 ; *********** outros LD *************
 LD (nn),A
  32H
  LOW nn
  HIGH nn
 LD (nn),HL
  22H
  LOW nn
  HIGH nn
 LD (nn),dd
  0EDH
  43H+dd
  LOW nn
  HIGH nn
 LD (nn),ix
  ix
  22H
  LOW nn
  HIGH nn
 ; **************
```

```
calc (ix d)
 ix
 calc+30H
 d
calc rega
 rega+calc
calc (HL)
 48+calc
PUSH qq
 0C5H+qq
PUSH ix
 ix
 0E5H
POP qq
 0C1H+qq
POP ix
 ix
 0E1H
EX DE,HL
 0EBH
EX AF,AF'
 08H
EXX
 0D9H
EX (SP),HL
 0E3H
EX (SP),ix
 ix
 0E3H
JR cjr,adrs   ;'$' representa o contador de programa.
 cjr
 adrs-2-$
JR adrs
 18H
 adrs-2-$
DJNZ adrs
 10H
 adrs-$-2
CALL cc,nn
 cc+0C4H
 LOW nn
 HIGH nn
CALL nn
 0CDH
 LOW nn
 HIGH nn
RET
 0C9H
RET cc
 cc+0C0H
CPL
 2FH
```

```
CCF
 3FH
NOP
 00H
RLCA
 7H
RLA
 17H
RRCA
 0FH
RRA
 1FH
SCF
 37H
RST p
 0C7H+p
ADD HL,dd
 9H+dd
ADC HL,dd
 0EDH
 4AH+dd
SBC HL,dd
 0EDH
 42H+dd
DAA
 27H
HALT
 76H
DI
 0F3H
EI
 0FBH
IM 0
 0EDH
 46H
IM 1
 0EDH
 56H
IM 2
 0EDH
 5EH
ADD IX,pp
 ODDH
 49H+pp
ADD IY,rr
 0FDH
 9H+rr
INC dd
 3H+dd
INC ix
 ix
 23H
```

```
DEC dd
 0BH+dd
DEC ix
 ix
 2BH
seta bit,(HL)
 0CBH
 seta+bit+6
seta bit,regb
 0CBH
 seta+bit+regb
seta bit,(ix d)
 ix
 0CBH
 d
 seta+bit+6H
blt
 0EDH
 blt
rot (HL)
 0CBH
 rot+6
rot regb
 0CBH
 rot+regb
rot (ix d)
 ix
 0CBH
 d
 rot+6
```

6. CONCLUSÕES

A utilização do meta assembler pelo grupo de desenvolvimento aumentou as possibilidades de utilização dos recursos de hardware disponíveis.

Estão sendo implementados em LDA montadores para os microprocessadores MC68000, 80286 e o microcomputador Z8.

Está em estudo a viabilidade de definição de uma linguagem assembler padronizada para vários microprocessadores, e a implementação de uma extensão da LDA que permita escrever os programas em uma linguagem estruturada do tipo PL/M e PL/Z.

- APÊNDICE

SINTAXE DA LINGUAGEM LDA: Linguagem de Descrição de Assembler

O formalismo BNF (Bankus Naur Form) é utilizado a seguir para descrever a sintaxe de presente linguagem.

```
< definição de assembler > ::= < declaração >
                                  <definição de assembler>

  declaração    ::= < declaração de nome de pseudo-opcode > |
                    < declaração de tabela de opcode > |
                    < declaração de tabela de constante > |
                    < declaração de limite numérico > |
                    < declaração de tamanho de palavra > |
                    < declaração de tamanho de endereçamento > |
                    < declaração de sintaxe > |
                    < comentário >

< declaração de nome de pseudo-opcode > ::=
                                pseudo < pseudo > = < labelpseudo >

< pseudo > ::= EQU | SET | ORG | PHASE | DEPHASE |
               DS1 | DS2 | DS4 | DEFB1 | DEFB2 | DEFB4 | DIW

< labelpseudo > ::= <label >

< declaração de tabela de opcode > ::= opcode < labelopcode > :
                                        < lista de opcodes >

< labelopcode > ::= <label minúsculo >

< lista de opcodes > ::=
            < tabulador > < nomeopcode > = < expressão >
               < lista de opcodes >

< nomeopcode > ::= < label >

< declaração de tabel de constante > ::=
              < label tabela de constante > :
                 < lista de constantes >

< label tabela de constante > ::= < label minúsculo >

< lista de constantes > ::=
           < tabulador > < nome constante > = <expressão >
               <lista de constantes >

< nome constante > ::= < label >
```

```
< declaração de limite numérico >  ::= range   < labelrange >  =
                 < expressão >< tabulador > .. < tabulador>< expressão >

< labelrange >    ::=  < label minúsculo >

< declaração de tamanho de palavra >  ::= word = < expressão >

< declaração de sintaxe> ::=< nome da operação> <tabulador><operando >

< nome da operação >  ::=  < labelopcode>   |  < label >

< operando>  ::=  (  < label tabela de constante >  ,
                     < labelrange > ,
                     < símbolo maiúsculo> ,
                     < símbolo decimal > ,
                     < outros símbolos > ) [ < operando > ]

< expressão de montagem >  ::=  <tabulador>< expressão >
                                  < expressão de montagem >

< comentário > ::=  ( ; | * )[<comentário>]

< expressão > ::=  <operação unária >  < expressão > |
                   <expressão > < operação > <expressão > |
                   <constante decimal > |
                   <constante hexadecimal > |
                   <constante octal > |
                   <constante binária > |
                   <constante alfanumérica > |
                   <label > :
                   <pc >

<pc >  ::= $

<constante alfanumérica >  ::= ( ' , " )  <string > ( ' , " )

<string >  ::=  <símbolo maiúsculo > |
                <símbolo minúsculo > |
                <símbolo decimal  > |
                <outros símbolos> |
                ''' | """ | ;
                  [ <string> ]

<constante decimal>   ::= < símbolo decimal>

<constante hexadecimal >  ::= $ < símbolo hexa > |
                                < símbolo hexa > H

<constante octal >   ::= < símbolo octal >  (O, Q)

<constante binária >   ::= <símbolo binário >  B
```

<label > ::= (<símbolo maiúsculo > , <caracteres de separação >)
[(<label > , $)]

<label maiúsculo > ::=
(<símbolo minúsculo > , < caracteres de separação >)
[(<label maiúsculo > , $)]

<operação unária > ::= - | NOT | LOW | HIGH | LOWW | HIGHW

<operação > ::= * | / | + | - | MOD | SHR | SHL | AND | OR | XOR

<símbolo maiúsculo > ::=
A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<símbolo minúsculo > ::=
a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

<símbolo hexa> ::=
(A|B|C|D|E|F | <símbolo decimal>)| [<símbolo hexa>]

<símbolo octal> ::= (0|1|2|3|4|5|6|70) [<símbolo octal >]

<símbolo decimal > ::= (0|1|2|3|4|5|6|8|9) [< símbolo decimal>]

<símbolo binário > ::= 0|1 | <símbolo binário >

<caracteres de separação > ::= . | _

<tabulador ::= espaço | tabulador |[<tabulador>]

<outros símbolos > ::=
* | (|) | [|] | | # | . | _ | ~ | | & | ! | : |
+ | $ | % | ' | | ? | ^ | | <tabulador>

- **BIBLIOGRAFIA**

1 _____ **8080/8085 Assembly Language Programming,** Santa Clara CA, Intel, 1979.

2 _____ **A Programmer's Guide to the Z8 Microcomputer** Application Note, Cupertino CA, Zilog, October 1980

3 _____ **Microcomputer Catalog** Databook, Mountain View CA, NEC Eletronics, 1984.

4 _____ **Series 32000** Databook, Santa Clara CA, National Semiconductor Corporation, 1984.

5 _____ **Utility Software Manual** MACRO-80, Microsoft, 1978.

6 _____ **Z8 Microcomputer Technical Manual,** Campbell CA, Zilog, April 1983.

7 _____ **Z80 CPU Programmer's Reference Guide,** Campbell Ca, Zilog, October 1982.

8 _____ **Z80 CPU Z80A CPU Technical Manual,** Cupertino CA, Zilog, September 1978.

9 AHO, Alfred V.; ULLMAN, Jeffrey D. **Principles of Compiler Design,** Reading MA, Addison-Wesley, April 1979.

10 BARRON, D. W. **Assemblers and Loaders,** London, McDonald, 1972.

11 KANE, Gerry; HAWKINS, Doug & LEVENTHAL, Lance. **68000 Assembly Language Programming,** Berkeley CA, OSBORNE/McGraw-Hill, 1981.

12 CALINGAERT, Peter **Assemblers, Compilers, and Program Translation** Potomac, Computer Science Press, 1979.

GRUPO V

INFORMATICA Y EDUCACION EN LA EMPRESA: UN DESAFIO A RESOLVER EN CONJUNTO

Horacio Eduardo Antonelli
Héctor Leónidas Sarasola
Universidad Católica de Córdoba - Banco Social de Córdoba
Córdoba - *Argentina*

NATURALEZA

El presente Artículo trata, al mismo tiempo, del Proyecto en curso y de la Experiencia realizada y evaluada hasta el momento, correspondiente a la primera fase de este Proyecto.

NIVEL

Apunta a la formación permanente de adultos e incluye el nivel de posgrado. Los educandos son adultos cuyas edades van desde los 20 a los 55 años, con estudios secundarios, muchos de ellos graduados en distintas profesiones.

OBJETIVO

El **objetivo general** del Proyecto es alcanzar una formación general en informática que permita a todo el personal del Banco, en sus respectivos niveles y funciones, abordar debidamente los problemas informáticos. (Ver objetivos del Proyecto en el Anexo Documentación).

El incontenible impacto de la tecnología informática en las empresas exige de éstas una rápida adaptación a la nueva forma sistémica de administración. Esto significa que debe hacerse un esfuerzo singular de educación para incorporar los conceptos e instrumentos de esta tecnología a través del hombre y no, como por lo general y equivocadamente se hace, a través de la máquina.

Esta tarea sólo puede llevarse a cabo haciendo coincidir tres factores indispensables: el compromiso del personal superior de la empresa, el aporte de las áreas de informática y la guía y conducción académica de una institución de prestigio en el ámbito educacional.

El presente Artículo pretende destacar que la solución concertada ofrece resultados, en las áreas de la Educación y de la Informática, que trascienden el mero ámbito computacional, ingresando en el campo de la Administración y llegando a modificar cursos de razonamiento tradicionales.

AMBITO

El Proyecto se lleva a cabo en el Centro de Perfeccionamiento en Administración de Empresas (CEPADE), dependiente del Instituto de Ciencias de la Administración de la Universidad Católica de Córdoba.

EVALUACION

La metodología fue especialmente desarrollada para permitir la evaluación de las experiencias educativas desde la perspectiva de los tres factores mencionados que intervienen en la misma: el personal de la empresa, las áreas de informática y la institución académica.

CURRICULA

En lugar de ajustar las experiencias proyectadas a la currícula se adaptó ésta a las primeras.

Efectivamente, en virtud de tratarse de un Proyecto a la medida de las reales necesidades de educación en informática del Banco, no se estimó conveniente utilizar ninguno de los programas existentes en la Universidad, sino que se decidió elaborar una currícula ad hoc.

ESTRUCTURA

La estructura del Proyecto en curso comprende tres fases que se corresponden con los tres niveles de personal del Banco Social de Córdoba: de Dirección, de Mandos Medios y de Ejecución. Estas fases, a su vez, se desagregan en 28 etapas y 121 actividades.

La duración estimada del Proyecto es de 1.269 horas, distribuídas de la siguiente manera:

- fase 1 = 422 horas;
- fase 2 = 600 horas;
- fase 3 = 247 horas.

AUTORES
Antecedentes Curriculares (Resumidos)

HORACIO EDUARDO ANTONELLI
Ingeniero Eléctrico Electrónico - Universidad Católica de Córdoba
Analista de Sistemas - Universidad Tecnológica Nacional
Ha participado en Congresos, Seminarios y Cursos de Perfeccionamiento en América y Europa
Jefe del Area Informática y Miembro del Consejo de Profesores del Instituto de Ciencias de la Admistración - Univ. Católica de Córdoba
Profesor de la Carrera de Ingeniería de Sistemas de la Universidad Tecnológica Nacional - Facultad Regional Córdoba
Ex-Director del Departamento de Sistemas de la Universidad Tecnológica Nacional - Facultad Regional Córdoba
Asesor de la Dirección de Informática de la Provincia de Córdoba
Desarrolló su actividad profesional en FIAT Concord SAIC y en PROCEDA S.A.
Corresponsal en Córdoba de la Red RCII de UNESCO
Consultor de empresas en el área de Informática.

HECTOR LEONIDAS SARASOLA
Bachillerato Humanista - Colegio Nacional Justo José de Urquiza
Especialista en Informática
Ha realizado numerosos cursos y seminarios sobre temas de su especialidad: análisis de sistemas, programación, computación y administración de áreas informáticas
Participó del Seminario Internacional sobre la Implementación de un Programa de Formación en la Empresa
Jefe de Trabajos Prácticos de la Cátedra de Procesamiento de Datos de la Facultad de Ciencias Económicas y de Administración - Universidad Católica de Córdoba
Miembro Titular del Primer Plenario Nacional de Informática de 1975
Representante del Banco Social de Córdoba ante la Asociación de Banco de Provincias (ABAPRA) para el tema Centros de Cómputos
Integrante del Comité de Capacitación de los Plenarios de Centros de Cómputos de ABAPRA
Integrante de la Comisión de Capacitación del Banco Social de Córdoba
Miembro Titular del Consejo Consultivo de Informática de la Provincia de Córdoba
Subgerente Departamental de Computación del Banco Social de Córdoba.

PROYECTO DE FORMACION EN INFORMATICA (PFI)

GENESIS

Situación Previa

Hacia fines de 1984 el Banco Social de Córdoba estaba desarrollando un Proyecto de Automatización de todas sus áreas. A medida que éste avanzaba se hacia evidente que las personas y la tecnología se separaban cada vez mas, aún cuando tenían una creciente interacción.

Se incorporaba tecnología, que llevaba ínsita nuevas ideas de organización y administración desconocidas por el personal. Se producía así una brecha conceptual cada vez mas peligrosa en sus consecuencias y, a pesar de que cada día había mas máquinas en uso, cada vez se comprendían menos sus implicancias sustanciales.

Demanda Educativa

Este diagnóstico de situación fue llevado al seno de la Comisión de Capacitación del Banco - cuyos integrantes pertenecen a su nivel gerencial - la que resolvió acudir al CEPADE en demanda, fundamentalmente, de un aporte de tecnología educativa en informática.

A efectos de caracterizar debidamente esta demanda, el Banco formuló un conjunto de requerimientos, en virtud de los cuales el **proceso educativo** debía:

- respetar la idiosincracia del Banco, en su doble naturaleza de institución bancaria y empresa del Sector Público Provincial;
- llegar a todos los empleados del Banco, dado que las técnicas informáticas afectan a todos sus niveles;
- estructurarse por niveles funcionales, respondiendo a la problemática específica de cada uno de ellos y las características de las personas que los integran;
- lograr una disminución del estado de ansiedad provocado por las confusas expectativas originadas en la incorporación de la tecnología informática;

- establecer un canal de comunicación que permita al personal no informático participar activamente en el proceso de informatización del Banco;
- enfatizar los aspectos conceptuales, de forma tal que los aspectos computacionales tengan un tratamiento complementario;
- transmitir los nuevos conceptos de organización y gestión que introduce la tecnología informática;
- asegurar el efecto "multiplicador" de la capacidad intelectual del hombre, ya que ésta es la consecuencia mas importante de la adecuada utilización de la computadora;
- tener en cuenta que una de las finalidades mas importantes de la inserción de la computadora en la empresa es la de liberar al hombre de las tareas rutinarias, para que pueda dedicarse prioritariamente a las creativas;
- posibilitar una participación activa del Banco en el Proyecto de capacitación, lo que implica que sus instructores deberán ser especialmente entrenados por la Universidad para trabajar en conjunto con los profesores del CEPADE.

Antecedentes del Proyecto

A fin de que diseñasen el Proyecto educativo que habría de responder a la demanda así caracterizada, la Universidad y el Banco designaron a sus respectivos **coordinadores académicos** - los autores del presente Artículo - quienes como primera medida elaboraron el borrador con los diversos **objetivos** a satisfacer: institucionales, generales y a los niveles de dirección, mandos medios y ejecución.

Una vez aprobados estos objetivos por la Comisión de Capacitación del Banco, los coordinadores prepararon un **anteproyecto**, estructurado por fases, que contemplaba diferentes alternativas de solución.

Este anteproyecto fue tratado por la Comisión de Capacitación y el CEPADE. De la concertación entre ambas partes surge el **Proyecto de Formación en Informática (PFI)**.

ESTRUCTURA

La estructura del PFI comprende tres **fases**, desagregadas en diversas **etapas**. Cada una éstas, a su vez, abarca distintas **actividades**. Para cada actividad se consigna el **responsable** de su ejecución, la **duración** estimada y la **salida** resultante.

Cada una de las tres fases está dirigida a cada uno de los tres grandes niveles de personal del Banco.

La **fase 1**, para niveles gerenciales, abarca dos seminarios: uno básico y otro avanzado, cada uno de una semana de duración a tiempo completo. Su dictado está a cargo de profesores del CEPADE.

La **fase 2**, para niveles medios, consta de un solo curso de una semana de duración a tiempo completo. El mismo se repetirá 12 veces y será dictado en conjunto por docentes del CEPADE e instructores internos del Banco, especialmente entrenados por la Universidad.

La **fase 3**, para niveles de ejecución, comprende un único minicurso de 6 horas de duración, que será preparado por el CEPADE y dictado por los mismos instructores internos del Banco. Este minicurso se repetirá las veces que sea necesario para capacitar a la totalidad de los auxiliares del Banco.

En el Anexo Documentación se adjunta la **estructura** integral del PFI, que incluye las etapas ya concluídas del mismo, es decir, de la experiencia realizada y evaluada hasta el momento.

METODOLOGIA

En primer lugar, se estableció como modus operandi la consulta y concertación permanente de las propuestas educativas del CEPADE con la Comisión de Capacitación del Banco Social de Córdoba.

Como pautas metodológicas el CEPADE planteó a esta Comisión el análisis de diversas alternativas de acción como respuesta a la demanda educativa formulada. La primera consistía en el desarrollo de los cursos de caracter general que el CEPADE dicta habitualmente. La segunda contemplaba la adecuación de estos cursos a las particularidades del Banco. La tercera comprendía el diseño de cursos "a medida", basados en una investigación previa de las necesidades generales y específicas de cada nivel del Banco.

Analizadas las tres alternativas, se optó por la última de ellas, en virtud de las evidentes ventajas que presentaba.

A continuación se ajustaron tanto el objetivo general del Proyecto como sus objetivos específicos - a nivel de Dirección, de Mandos Medios y de Ejecución - a los que se les hizo corresponder las tres fases del Proyecto con sus respectivos marcos teóricos y contenidos sintéticos.

Además, se concertaron las condiciones operativas, a fin de satisfacer los requerimientos pedagógicos y didácticos de la experiencia educativa.

Asimismo, para cada Fase se definió como primera etapa el diagnóstico de las necesidades de capacitación del nivel correspondiente. Para realizar este diagnóstico se decidió efectuar una investigación con muestras representativas de los futuros participantes.

En función de las necesidades de capacitación detectadas se ajustó la metodología a utilizar.

EXPERIENCIA REALIZADA

DIAGNOSTICO

La Experiencia concluída y evaluada hasta el momento - que consiste en el desarrollo del **curso básico** para Directivos - se puso en marcha en el tercer trimestre del año 1985, con la elaboración del **cuestionario de relevamiento** adjunto en el Anexo Documentación. Esta tarea fue realizada por especialistas del CEPADE, tomando como base los datos obtenidos en contactos previos entre los coordinadores académicos y en las reuniones mantenidas con la Comisión de Capacitación.

Se contó, además, con la asistencia profesional de la Responsable de la División Capacitación del Banco, quién aportó su juicio técnico desde la óptica particular inherente a este sector de la Institución.

A partir de los resultados de la **encuesta** realizada, que recogió las opiniones de todo el nivel gerencial del Banco, se prepararon y efectuaron una serie de **entrevistas personales** que alcanzaron al 40% de los gerentes.

El **informe de diagnóstico** resultante, una vez aceptado por la Comisión de Capacitación, sirvió de fundamento para el diseño del **curso básico** y la preparación del **material** correspondiente, que incluye un glosario y un manual de referencia para directivos.

DISEÑO

La etapa de diseño, basada en el informe de diagnóstico y en los objetivos específicos predefinidos, comprendió:

- la determinación de los **contenidos analíticos** y su adecuación, en amplitud y profundidad, a partir de los contenidos sintéticos;
- la cuidadosa preparación de los **casos de estudio** para que reflejasen fielmente la problemática del Banco (Ver Anexo Documentación);
- la elaboración de la **guía didáctica** con explicitación de los contenidos y actividades de cada unidad didáctica;
- la preparación de una **guía metodológica** para el correcto ajuste de la metodología específica a utilizar en cada unidad didáctica.

DICTADO

Para facilitar el desarrollo del curso básico, se concertaron previamente **condiciones operativas** que pueden consultarse en la Anexo Documentación.

El curso tuvo lugar en la sede del CEPADE, durante una semana completa y en el horario de trabajo del Banco, por lo que su dictado debió desdoblarse.

Al iniciar el dictado de cada curso el docente, de común acuerdo con los participantes, precisó qué **expectativas** podrían ser satisfechas y cuáles no, fijando claramente los límites de la problemática a tratar y ajustando consecuentemente la guía temática.

En esta oportunidad se puso de manifiesto el elevado nivel de **motivación** logrado en los participantes, aún entre los mas renuentes a la inserción de la informática en el Banco, consecuencia ésta de la aplicación de la metodología de realización de encuestas y entrevistas previas.

Esta motivación se tradujo en una significativa **participación** de los asistentes en el desarrollo de las clases, la que se vió facilitada por los casos de estudio que reflejan la problemática habitual de cada uno de ellos y de su entorno.

EVALUACION

Para llevar a cabo el proceso de evaluación se establecieron dos enfoques complementarios: uno desde el punto de vista de los participantes y otro desde la óptica de los profesores y los coordinadores académicos.

En el primer caso, habida cuenta del elevado grado de comunicación que se logró en el proceso de enseñanza-aprendizaje, se consideró conveniente y oportuno desestimar el cuestionario escrito previsto en el Proyecto, para permitir a los participantes expresarse con mayor fluidez.

La evaluación, en consecuencia, se realizó en forma pública y verbal, exponiendo cada uno de los participantes sus **opiniones** respecto del nivel de satisfacción alcanzado con relación a sus **expectativas** personales.

Las expectativas y opiniones de los participantes fueron registradas y posteriormente incorporadas en el **informe de resultados** que el CEPADE remitió al Banco y que se adjunta en el Anexo Documentación.

Este informe de resultados se completó con la **evaluación de los docentes y los coordinadores académicos**, quienes presentaron conclusiones positivas, fundamentadas en el elevado grado de satisfacción de los objetivos propuestos.

CONCLUSIONES

Los resultados de la experiencia realizada hasta el momento son consecuencia de la integración de la teoría con la práctica, a través de la aplicación de técnicas de investigación e innovación educativa y, fundamentalmente, de la concreción de las ideas de:
planificación educativa a mediano y largo plazo - complementación institucional - concertación de objetivos - investigación de necesidades - informática como objeto y como medio del aprendizaje - comunicación docente-alumno - motivación y participación - enseñanza-aprendizaje personalizados - evaluación permanente.

Estos resultados se evidencian en el manifiesto cambio de actitud de los participantes con relación al fenómeno informático, como así también en su predisposición para participar decididamente en las siguientes etapas del Proyecto.

Concluyendo, el hito alcanzado justifica la continuidad de este Proyecto encarado en conjunto por la Universidad Católica de Córdoba y el Banco Social de Córdoba.

QUIMANCHE: UN PROYECTO CHILENO PARA LA EDUCACION EN COMPUTACION

Ruth Donoso Villegas
Colaboradores: *Colette Hollemart V., Herminia Ochsenius A., Víctor Ochsenius E.*
Pontificia Universidad Católica de Chile
Chile

Quimanche es una palabra araucana que está formada por dos raíces, QUIMAN (gran saber) CHE (Hombre). Significa por tanto sabio o el hombre de gran saber.

Heros elegido este nombre para destacar que en esta incorporación masiva de la computación en educación, es esencial mantener las características propias de nuestra cultura. Y la tecnología ha de estar supeditada el hombre y al saber.

Por ello nuestro proyecto se centra de manera fundamental en el profesional de la enseñanza, el profesor, tanto en las etapas del ciclo básico como del ciclo medio. Este profesor es el que hará la labor fundamental de poner la tecnología al servicio del saber y del enseñar, y no permitirá que sea la educación y el hombre el que se ponga al servicio de la tecnología.

Así pues, fue natural que este proyecto naciera en la Universidad, la Pontificia Universidad Católica de Chile, y al amparo de un Plan de Perfeccionamiento de Profesores.

HISTORIA DEL PROGRAMA

En enero de 1982 se desarrolló la primera jornada de perfeccionamiento de profesores, donde se dictó un curso de Estadística, uno de Geometría y otro de uso de máquinas calculadoras programables.

Se calculó una asistencia de 40 alumnos por cursos. En todos ellos la matrícula superó los 60 alumnos y el curso de uso de máquinas calculadoras contó con 116 profesores alumnos.

Este fue el primer paso del Programa de Perfeccionamiento de Profesores.

Así comenzaron a surgir inquietudes:
- Se elaboró un programa para crear un postítulo como profesor en Ciencias Básicas. Se elaboraron los programas y planes de estudio; sin embargo, éste no se materializó porque no pudimos integrar al área de Biología, quienes tenían otros intereses inmediatos.

- Se formaron grupos de investigación en educación y computación y se creó software en matemáticas.

- Se comenzaron a desarrollar seminarios de computación y educación para los profesores del equipo de trabajo.

- Se hicieron los contactos con profesores, sicopedagogos y otros profesionales que estaban produciendo softeducativo o trabajando en educación con micro - computadores.

ESTADO ACTUAL DEL PROGRAMA

En las seis jornadas de perfeccionamiento que se han desarrollado desde enero de 1982 a la fecha, se han atendido a 1373 profesores - alumnos en 27 cursos distribuídos:

CURSOS	N°DE ALUMNOS
9 de Matemáticas	490
9 de Física	276
3 de Química	150
6 de Computación	457
Total 27 cursos	1373 alumnos

Debido a la gran demanda que en el último tiempo se ha tenido en lo que se refiere a computación educativa, el Programa de Perfeccionamiento de Profesores, hoy tiene dos tipos de actividades:
A: Todos los cursos o talleres de Matemáticas y Ciencias en general, y
B: Actividades en computación de acuerdo al proyecto QUIMANCHE.

ACTIVIDADES DE TIPO B.

Es una procupación actual del Programa de Perfeccionamiento de Profesores el aporte que puede hacer la computación al proceso enseñanza-aprendizaje, ya que se ha podido detectar en las últimas encuestas, el interés que los profesores tienen en esta área.

El gran problema que el profesor enfrenta con el auge del uso del computador como herramienta educativa; y la posible utilidad que este podría tener en dicho proceso, ha sido el móvil que originó el Proyecto QUIMANCHE, que describimos a continuación.

En forma muy suscinta, el proyecto consiste en:
1.- Crear un centro de perfeccionamiento en computación aplicada a la educación.

2.- Crear softwear educacional en español, aplicados y aplicables a la realidad nacional y de acuerdo a los programas educacionales vigentes.

La particularidad de este proyecto está en que el profesor universitario o investigador no pasa a reemplazar al profesor de la escuela o liceo, antes bien, pone énfasis en la importancia de éste para transmitir los conocimientos a sus alumnos. De este modo el medio del niño no se ve alterado, siempre es su profesor el que esta presente.

No así muchas otras experiencias en que el investigador de la Universidad va à la Escuela e imprime su sello. Nosotros preparamos al profesor para que él siga siendo el que enseña o el que aprende del niño. Pero es él, el responsable el que cumple su rol, nosotros conocemos su experiencia y le ayudamos a orientarla si es preciso. Tenemos confianza en el profesor porque conocemos su quehacer, es un profesional que sabe lo que le corresponde hacer.

Por eso es que los objetivos generales y específicos del proyecto son los siguientes.

OBJETIVOS GENERALES.
Crear un Centro de Perfeccionamiento en Computación educativa a nivel nacional, con proyecciones al ámbito latinoamericano, con el propósito de:
- Perfeccionar profesores de enseñanza básica y media en computación educativa.

- Crear, desarrollar y adaptar tenología educacional en computación, acorde a la realidad nacional.

OBJETIVOS ESPECIFICOS.
a) Ofrecer cursos de capacitación y perfeccionamiento a los profesores de enseñanza básica y media.

b) Proporcionar los elementos necesarios para que el profesor haga un buen uso del computador en la educación.

c) Encauzar los conocimientos didácticos y pedagógicos que el profesor tiene, en un uso apropiado a través del computador.

d) Dar al profesor una herramienta eficaz para ampliar los conocimientos de sus alumnos en el área que enseña.

e) Enseñar lenguaje computacional acorde con las necesidades educacionales.

f) Formar grupos interdisciplinarios con el fin de desarrollar experiencias e investigaciones en los colegios y en la Universidad.

g) Colaborar con los centros de formación de profesores en el diseño de los curriculum y la implementación de los programas en el área computacional.

h) Reunir todos los esfuerzos en esta línea, que se encuentren dispersos en el país (investigadores y memoristas).

i) Crear y adaptar software educacional basado en los programas de enseñanza básica y media actualmente vigentes, para ser utilizados por los profesores como apoyo a la docencia.

j) Formar centros regionales y comunales apoyados por empresa del área.

Como consecuencia, se han desarrollado las siguientes actividades:

1) Se han dictado seis cursos de computación a más de cuatrocientos cincuenta profesores-alumnos en tres de las seis jornadas realizadas.

2) De los profesores que han hecho cursos de computación un 80% ha desarrollado talleres con sus alumnos o colegas en sus respectivos colegios.

3) De los profesores que toman cursos de computación, un gran número continua perfeccionándose y plantean desarrollar experiencias e investigaciones.

4) El primer semestre del año 1985, con la asistencia de 210 profesores de enseñanza básica y media, se desarrolló un seminario de lenguaje LOGO.

5) El segundo semestre 1985, se hizo un seminario de lenguaje LOGO para profesores de la Pontificia Universidad Católica de Chile.

6) Se realizó el Primer encuentro Nacional de Experiencias y Posibilidades de la Computación en la Educación (Diciembre de 1985) con la asistencia de cuatrocientos profesores de Enseñanza Básica y Media.

7) Se desarrolló la Sexta Jornada de Perfeccionamiento de Profesores a la que asistieron 312 profesores-alumnos, de Arica a Punta Arenas (Enero de 1986).

8) Se realizó un estudio exploratorio (Febrero 1986) en tres colegios de Santiago, (dos de ellos en sectores marginales), para investigar tres métodos de la enseñanza del lenguaje LOGO.*

9) Se están dictando dos cursos de computación a 95 profesores de distintas asignaturas y niveles, durante los días sábados, habiendo registrado una demanda de 200 profesores-alumnos. Esta reducción se debe a la falta de medios para su implementación, pese a contar con los equipos necesarios.

10) Se está realizando en la Facultad de Matemática un seminario de PROLOG para profesores de la Pontificia Universidad Católica de Chile y otras universidades del país con la asistencia de profesores invitados.

11) En Murcia, España, durante el 1° Semestre del año, realizó un perfeccionamiento en computación educativa, una Profesora de la Facultad de Matemática, becada a través del Proyecto Quimanche.

12) Se realizó un seminario para profesores de enseñanza básica y parvularia "Desde el Jardín Infantil a los sistemas de Información", el que fue conducido por el profesor Ingeniero Raúl Dorfman enviadao por CREI con la participación de 40 profesoras parvularias; del cual estan saliendo otras experiencias.

ACTIVIDADES POR DESARROLLAR

1) Coordinar las actividades de las sedes que trabajarían con el Proyecto Quimanche, haciendo un convenio de aportes y compromisos.

   Estas sedes son: Arica e Iquique dependiente de la Universidad de Tarapacá; Antofagasta y Coquimbo dependiente de la Universidad del Norte; Temuco y Talcahuano sedes de la Pontificia Universidad Católica de Chile; Concepción con la Universidad del Bío- Bío, Valdivia con la Universidad Austral y Punta Arenas con la comporación de Colegios Salesianos.

2) Apoyar la participación de colegios del Area Metropolitana en investigaciones que se están desarrollando, tales como:

   a) Liceo Juana de Ibarbouron "La computación enseña a aprender".

   b) Centro Educacional Omega "Integración de personas de tercera edad y niños, jugando con el computador".

* Comunicación Enviada.

c) COMUDEF (Corporación Municipal de la Florida), Liceo Los Almendros "Exploración de un método de enseñanza del lenguaje LOGO" (Réplica de la experiencia de Febrero).

d) COMUDEF, Liceo Andrés Bello, "Exploración de un método de enseñanza del Lenguaje LOGO" (Réplica de la Experiencia de Febrero)

e) COMUDEF, Seminario para profesores de niños con problemas de aprendizaje, con la utilización del lenguaje LOGO.

3) Planificar y realizar el Segundo Encuentro de Experiencia y Posibilidades de la Computación en la Educación, a realizarse entre el 15 y el 17 de Septiembre.

4) Participar en charlas, exposiciones y simposium dentro y fuera del país dando a conocer el programa.

5) Planificar la séptima Jornada de Perfeccionamiento de Profesores, a realizarse en enero de 1987.

Estas actividades realizadas por el Programa de Perfeccionamiento de Profesores están diseñadas para que de ellas surja en forma natural el Proyecto Quimanche

Este Proyecto, una vez aprobado en forma definitiva, se desarrollará en las siguientes etapas:

1° Preparar el personal docente del proyecto y formar grupos interdisciplinarios:
a) profesores que impartirán los cursos de perfeccionamiento

b) profesores jefes de talleres

c) profesores ayudantes de los cursos.

2° Adiestramiento de profesores de educación básica y media, en la Región Metropolitana y en los centros regionales implementados.

3° Adiestramiento de profesores de enseñanza básica; continuación del entrenamiento de enseñanza media; recopilación de software que se ha generado.

4° Continuación de lo anterior.

Desarrollo de software en español para las distintas asignaturas.

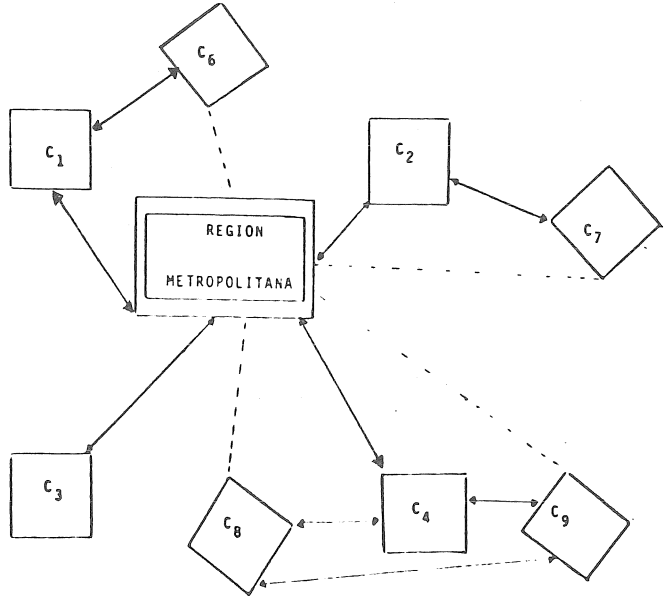
FUNCIONAMIENTO DEL PROYECTO QUIMANCHE A LO LARGO DEL PAIS.

Por las características propias de la geografía chilena el Proyecto Quimanche ha dividido el país en cuatro zonas: Zona Norte, Zona Sur, Zona Austral y Región Metropolitana.

En cada Zona se contemplan varios centros coordinados entre si y con la Región Metropolitana. En ellos el Proyecto se desarrollará ligado, en lo posible, a sedes de la Universidad ejecutora u otras Universidades que se integren a este programa. Es así como en la Zona Norte se formarían centros en Arica, Iquique, (Universidad de Tarapacá), Antofagasta y Coquimbo (Universidad del Norte) en la Zona Sur se tiene Talcahuano, Concepción y Temuco (Sedes de la Pontificia Universidad Católica de Chile). En Punta Arenas está trabajando la Corporación de Colegios Salesianos en la línea del Proyecto Quimanche. Y en la Región Metropolitana estaría el Centro General, formándose ya otros en colegios y Municipalidades.

Para cada Zona se contempla un crecimiento relativamente independiente de modo que pueda adoptar las líneas del Proyecto a la realidad socioculturales y económica de las diversas regiones y provincias. Pero el tipo de capacitación, los cursos que se dictan, sus programas y la filosofía central han de ser las mismas para todo el país. Esto se logra a través de una estrecha interrelación con el Centro General, quien recibe y transmite información, coordina la labor, y principalmente forma a los monitores que enseñarán en la Zona.

Esto se visualiza en el siguiente esquema:



CURSO INTRODUCTORIO DE COMPUTACION CON MUY ESCASOS RECURSOS. UNA EXPERIENCIA

Juan V. Echague
Facultad de Ingeniería
Montevideo - *Uruguay*

RESUMEN

Se relata la experiencia de un curso introductorio a la informática en carreras universitarias de formación de profesionales en esa área, dictado con muy escasos recursos.

El curso se organiza alrededor de la programación como resolución de problemas. La atención se centra en la especificación, la programación imperativa y la interacción de estos dos elementos. Se utiliza como lenguaje de especificación la lógica (cláusulas de Horn), y como lenguaje de programación un subconjunto de Pascal.

Fue dictado durante el semestre de otoño de 1986 por el Departamento de Programación del Instituto de Computación (Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay).

1) Antecedentes.

La Universidad de la República ofrece dos carreras en informática: 'Ingeniero de Sistemas en Computación' y 'Analista Programador', ambas por medio de la Facultad de Ingeniería.

Fueron creadas en el año 1974, al inicio del período de intervención universitaria impuesta por el gobierno de facto. Los legítimos gobernantes de la Universidad no pudieron discutir ni decidir sobre los presupuestos básicos, los títulos ni las currícula de estas carreras desde su creación hasta el cese de la intervención, en 1984.

En 1985 la Comisión de Area de Computación eleva al Consejo de Facultad de Ingeniería una propuesta de modificaciones de contenidos de materias a fin de lograr 'una rápida actualización' de estos (CIn85, CAC85). Esa propuesta es aprobada.

Introducción a la Computación es una materia semestral, ubicada al inicio de las carreras en informática. En el año 1986 se dicta por vez primera como tarea del Departamento de Programación del Instituto de Computación, con el contenido propuesto y aprobado por el gobierno universitario legítimo.

La Comisión de Area de Computación (CAC85) recomienda para ella:

'Objetivos del curso:
- Introducir las nociones básicas de lógica.
- Introducir a la metodología de especificar y resolver problemas (de procesamiento de datos en particular).
- Introducir al diseño de algoritmos simples.
- Introducir un lenguaje de alto nivel (Pascal).'

2) Situación material del dictado del curso 1986.

La matrícula en el curso fue de 1248 estudiantes. Lo que mantiene la tendencia de matrícula numerosa y creciente presente desde hace varios años.

El curso fue dictado por 3 (tres) docentes, que sumaban una dedicación semanal de 110 (ciento diez) horas.

La relación tiempo-docente/estudiante es de 3.8 minutos semanales para este curso (el promedio del Departamento fue en ese semestre de 9 minutos).

3) El curso.

3.1) Los mecanismos usados para tomar las decisiones.

Las decisiones fundamentales en cuanto a contenido y forma del curso reflejan consensos alcanzados en reuniones y seminarios del Departamento de Programación.

En las reuniones participaron docentes del Departamento de Programación y estudiantes avanzados. Su objetivo fue la preparación de todos los cursos del semestre de otoño de 1986, tanto en su contenido como en su forma. Se desarrollaron a partir de mediados de febrero hasta el inicio de los cursos, a principios de mayo.

Los seminarios fueron dirigidos por docentes del Departamento de Programación, con asistencia de docentes y estudiantes avanzados. Cubrieron los temas cláusulas de Horn y metodología de resolución de problemas. Sus objetivos fueron: 1) profundización en el tema, 2) detección de los puntos de mayor dificultad para los estudiantes y 3) ensayo de distintas maneras de introducir los conceptos.

3.2) El contenido del curso.

El contenido del curso se organizó alrededor de las nociones de especificación y programación, articuladas en el marco del tema 'resolver problemas'. Cada una de estas nociones se enfrentó, como veremos mas adelante desde un marco conceptual distinto.

Esta dualidad es propia del estado de la discusión e investigación en el área de programación. Es ineludible enfrentarla hoy en el estudio de la programación. Hacerlo lo antes posible fue una de las decisiones de diseño mas profundas de este curso.

En los trabajos de preparación se revisaron, como acercamientos posibles a la especificación, técnicas funcionales, algebraicas, lógicas y de pre-condición/pos-condición, además de la opción por defecto de especificar problemas en lenguaje natural (L&Z77).

Se optó por la especificación en lógica, en particular, utilizando cláusulas de Horn. La lógica formal es ampliamente aceptada como lenguaje de especificación en la ciencia de la computación. Se adapta de manera ideal a la representación del conocimiento y a la descripción de problemas, independiente de la elección de un lenguaje de programación (Kow84).

Otra razón, para nada menor, fue la disponibilidad imediata de un texto sobre el tema: 'Logic for problem solving' (Kow79), que no reclama conocimientos previos de lógica ni programación.

Además del estudio de los temas de lógica y especificación, el manejo de cláusulas de Horn como herramienta de trabajo fue muy fértil al permitir ejemplificar metodologías ascendentes y descendentes (Kow79) y, mediante ejercicios, presentar las nociones de recurrencia, decidibilidad, gramáticas y lenguajes.

Algunas técnicas de resolución de problemas ('path-finding' y 'and-or trees') tienen una representación muy directa en cláusulas de Horn (Kow79), por lo que este tema se enfrentó con este mismo esquema conceptual.

En cuanto a programación el objetivo planteado fue el estudio de bases conceptuales de algún paradigma de los lenguajes de programación, prestando principal atención a los aspectos 'semánticos', dejando en un lugar secundario los 'sintácticos'.

En la preparación del curso se revisaron como opciones los paradigmas de lenguages lógicos, funcionales e imperativos. Los argumentos a favor de los dos primeros se centraron en la claridad y el desarrollo alcanzados en ellos (Fur86), y la convicción que encierran, en alguna forma, los lenguajes de programación del futuro cercano.

Sin embargo se optó finalmente por estudiar los fundamentos de los lenguajes imperativos, entendiendo estos como aquellos cuya característica central es permitir la creación de variables (Hor84). Dos razones poderosas para esta decisión fueron la familiaridad del cuerpo docente con lenguajes de este tipo y su amplia difusión en el medio.

En el curso se estudió una máquina teórica, de arquitectura Von Neumann. Luego se introdujeron nociones de lenguajes de alto nivel imperativos (variable, asignación, tipo, secuencia, selección e iteración) como modelaciones, de 'mas alto nivel' del lenguaje absoluto de esa máquina. Se tomaron pequeños ejemplos de media docena de lenguajes de alto nivel, poniendo acento en las identidades 'semánticas' existentes detrás de las diferencias 'sintácticas'.

Se postergo tanto como fue posible, la introducción de la sintaxis de un lenguaje de programación en particular. Con esto buscamos minimizar el 'efecto de lengua materna' que tiene el primer lenguaje de programación que se aprende.

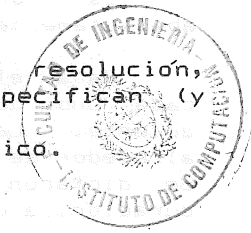
El lenguaje de programación utilizado fue una versión reducida de Pascal (J&W75). Soporta unicamente variables enteras, lógicas y de caracteres, sentencias de asignación, de selección (solamente 'if'), de iteración (solamente 'while' y 'repeat') y entrada/salida sobre archivos 'standard' (como sentencias). No se incluye reales, tipos estructurados ni subprogramas. En una sesión de alrededor de una hora se explicó el manejo de la cartilla sintáctica y se señalaron las correspondencias con la 'semántica' ya vista.

El difícil acople de los dos esquemas conceptuales fue también analizado en el curso teórico en varias oportunidades. Allí se pasó revista al proceso de resolución de problemas utilizando una computadora, los orígenes históricos de las nociones de especificación y programación imperativa, los 'distintos niveles de abstracción', su influencia en el costo de la actividad humana de programar y las posibles alternativas de futuro.

Podemos resumir el contenido del curso en el siguiente esquema:

Parte 1. Especificación.

Orientada a los problemas y las técnicas de resolución, con un 'alto nivel de abstracción'. Se especifican (y resuelven) problemas con lógica de cláusulas.
Ocupa el 60% del teórico y el 85% del práctico.



Parte 2. Programación.

Se presenta como actividad de resolución (posiblemente automatizable en parte) de problemas. Se discute el papel de la especificación y de un agente (quizas mecánico) que puede realizar acciones. Se presentan también las nociones de procedimiento y algoritmo.
Ocupa el 4% del teórico.

Parte 3. La máquina de Von Neumann.

Se estudia un modelo simplificado de computador con arquitectura Von Neumann. Memoria, unidad de proceso y conjunto de instrucciones. Es el punto de 'menor nivel de abstracción' del curso. Se trabaja alrededor de algunos problemas, comparando su especificación y un programa (en lenguaje absoluto de esa máquina) para resolverlo.
Ocupa el 9% del teórico.

Parte 4. Los lenguajes imperativos.

Se introducen como una forma mas cómoda de trabajo en una máquina Von Neumman, que permiten subir en el 'nivel de abstracción'. Se tratan las nociones de lenguajes de alto nivel, compilador, variable, expresión, asignación, tipo, secuencia, selección e iteración, con ejemplos en varios lenguajes de programación. El acento es puesto en las nociones de variable y asignación.
Ocupa el 18% del teórico.

Parte 5. Un lenguaje imperativo minimo.

Se introduce la sintaxis de un subconjunto reducido de Pascal, haciendo referencia a la semántica vista en la parte 4. Se muestran ejemplos.
Ocupa un 9% del teórico y un 15% del práctico.

3.3) La forma del curso.

La forma de dictado del curso estuvo determinada absolutamente por las condiciones materiales.

Para el dictado de teórico se trabajó en tres grupos, cada uno de alrededor de 400 estudiantes a cargo de un docente. La asistencia real fue (sumando los grupos) de alrededor de 600 estudiantes, estable a lo largo del curso. Se dictaron para cada grupo 30 clases de 2 horas de duración, a un ritmo de 3 clases semanales.

Los cursos prácticos estuvieron a cargo del gremio estudiantil (Centro de Estudiantes de Ingeniería). Se organizaron asi 33 grupos de práctico (llamados 'grupos de auto-estudio' por razones históricas) que se reunían 2 horas semanales a cargo de 135 estudiantes-colaboradores. Es muy difícil obtener datos sobre su funcionamiento, podemos evaluar que globalmente 400 estudiantes participaron.

Los ejercicios practicos (42 a lo largo de todo el curso) fueron propuestos por el Departamento como material de trabajo para esos grupos. Fueron planteados en sesiones de trabajo de estudiantes-colaboradores asistidos por un docente con una semana de antelación a su presentación en los 'grupos de auto-estudio'. Una resolución modelo era distribuida luego a los estudiantes-colaboradores.

Todos los ejercicios prácticos fueron 'de escritorio'. No hubo en ningún momento acceso de los estudiantes a computadoras.

Como bibliografía se empleó el texto de Kowalski ya citado (los cuatro primeros capítulos) para la primera parte del curso. Y un folleto de apuntes editado por la Facultad de Ingeniería (CET86) para el resto.

El curso fue completamente libre. No se controló asistencia ni se tomaron pruebas parciales.

5) Una primera evaluación.

A la fecha de escribir este trabajo no se conocen los resultados del primer examen de evaluación de este curso. Por lo tanto, solo pueden hacerse consideraciones subjetivas sobre esta experiencia.

Se cumplió globalmente lo propuesto en cuanto contenido del curso. La respuesta estudiantil ante esto fue variada, aunque podemos describir algunas pautas recurrentes.

La presentación de la lógica como herramienta de especificación de problemas fue aceptada con naturalidad por parte de los estudiantes, accediendo sin mayor dificultad a nociones de recursión, unificación, etc. El tema fue considerado globalmente 'fácil'. Hubo cierto desconcierto atribuible a la inadecuación del tema a las fantasías de los estudiantes sobre cuales son los temas 'importantes' en computación.

El tema de la programación imperativa y de un lenguaje de programación imperativo fue considerado por los estudiantes muy árido. Se lo valoraba como un tema muy importante. Muchos estudiantes complementaron el curso en el conocimiento del Pascal a traves de libros y manuales.

La disponibilidad de un texto para la primera parte fue muy importante. Tradicionalmente, en la carrera no existían 'textos del curso'. Eso, unido a que solo existe en versión inglesa, demoró el acceso de los estudiantes a él. Sin embargo, ante la inminencia del examen, el aprovechamiento del libro creció.

El tema programación imperativa tuvo varios problemas: no se dispuso de un texto adecuado, no se elaboraron prácticos sobre el tema ni se presentaron técnicas de resolución de problemas mediante algoritmos. Posiblemente el primero de los problemas fue el origen de los otros.

Fue claramente perceptible, sobre el fin del curso, el alto desgaste en los docentes.

6) Conclusiones.

En lo referente a cursos introductorios para carreras en informática, dictados en situaciones de muy escasos recursos, hay algunos elementos que esta experiencia contribuye a apoyar, sin ser concluyente al respecto.

Sería posible incluir herramientas de alto nivel de abstracción, con un aprovechamientos satisfactorio. Esto permitiría introducir facilmente conceptos fundamentales.

Sería posible incluir algunos elementos de la discusión académica en temas básicos del momento. Esto parece una posición necesaria en un terreno en que los cambios se procesan tan rapidamente.

Bibliografía.

CAC85. Resumen de cambios propuestos al plan de estudios de Ingeniería de Sistemas en Computación y Analista Programador. Comisión de Area de Computación, Facultad de Ingeniería. Julio de 1985.

CIn85. Informe de la Comisión de Informática al Consejo Interino de la Facultad de Ingeniería. Febrero de 1985.

CET86. Apuntes sobre programación imperativa. Crispino G., Echague J., Tasistro A. 1986. Publicación interna de la Facultad de Ingeniería.

Fur86. Paradigmas de lingugens de computaçao. Furtado. 1986. Editora da Unicamp.

Hor84. Fundamentals of Programming Languages (2nd edition). Horowitz, Ellis. 1984. Springer-Verlag Berlin Heidelberg New York Tokyo.

Kow79. Logic for problem solving. Kowalski, R. A. 1979. New York, Amsterdam: Elsevier.

Kow84. The relation between logic programming and logic specification. Kowalski, R. A. 1984. En 'Mathematical Logic and Programming Languages', editado por C.A.R. Hoare y J.C.Shepherdson, Prentice/Hall International.

J&W75. Pascal. User manual and Report. Jensen, Kathleen y Wirth, Niklaus.

L&Z77. An Introduction to formal specifications of data abstractions. Liskov, Barbara & Zilles, Stephen. 1977. En 'Current trends in programming methodology', volumen 1 'Software Specification and Design' editado por R. T. Yeh. Prentice - Hall Inc.

METODOS CONSTRUCTIVOS DE APRENDIZAJE

Elida Beatriz García Rozado
Javier Alcuri
Sergio Fixman
Alberto Savloff
Buenos Aires - *Argentina*

I.- I N T R O D U C C I O N.

El presente trabajo vuelca los resultados de una investigación realizada en el INSTITUTO NCR DE CIENCIAS DE LA COMPUTACION (Buenos Aires, Argentina), durante los años 1985-86, sobre las características que puede adquirir el proceso de aprendizaje utilizando computadoras y el lenguaje LOGO como herramientas.

Los hechos educativos que se analizan son: 1) La ENSEÑANZA LOGO, en cuatro cursos, con un promedio de 6 alumnos en cada uno, con edades comprendidas entre los 11 y los 18 años. 2) Una experiencia piloto consistente en un curso de GEOMETRIA CONSTRUCTIVA con dos alumnos de 14 y 17 años.

Los alumnos concurrieron tres veces por semana durante dos horas diarias (aunque normalmente se extendían en media hora y más, por su gran motivación), teniendo cada uno a su disposición un computador personal.

El lenguaje utilizado es el TLC-LOGO, marca registrada por The Lisp Company, bajo sistema operativo CP/M, corrido en computadoras personales NCR DECISION MATE V de 64 K de memoria.

II.- Q U E E S E L L E N G U A J E L O G O ?

LOGO es una voz derivada del griego "logos", empleada en el MIT (Massachusetts Institute of Technology), desde 1.970, por el equipo de Seymour Papert y Marvin Minsky para designar un **proyecto situado en el punto donde convergen las investigaciones sobre Inteligencia Artificial (IA) y ciencias de la educación.**

LOGO es una extensión algorítmica de LISP -lenguaje informático para el procesamiento de listas- sustentado por la moderna computación basada sobre objetos, patrones y patrones de manipulación y utilizado en los estudios de IA que se hallan orientados a la resolución de problemas.

LOGO es un sistema experto cuya aplicación no requiere conocimientos especializados por parte del usuario. Y, qué es un sistema experto? Es un sistema computacional basado en estudios de IA, que muestra un nivel de competencia en un campo particular del conocimiento: en este caso, **el aprendizaje**, el cual, si fuera evidenciado por un ser humano, sería considerado como un un comportamiento inteligente.

Pero LOGO es más que lo dicho anteriormente. Designa:
* una teoría del aprendizaje,
* un lenguaje de comunicación, y
* un entorno adecuado y gratificante,
todo lo cual, permite y posibilita proyectar luz sobre los procesos mentales a que recurre un individuo humano para resolver los problemas que se le plantean proponiendo una solución, mediante la cual incide -por su accionar- sobre el mundo exterior.

*** LOGO COMO TEORIA DEL APRENDIZAJE:**

Como tal se basa en la famosa teoría psicogenética-evolutiva de la inteligencia de JEAN PIAGET -del cual Papert es uno de sus discípulos-.

El gran biólogo, psicólogo y epistemólogo suizo muestra, a través de una extensa producción científica -pormenorizada y exhaustiva-, lo que los niños a diferentes edades pueden y no, aprender a realizar.

Ahora bien, una contribución importantísima de Piaget reside en que -a pesar de las "sorprendentes deficiencias (?) que evidencian los niños a ciertas edades" (1)- son ellos, por sí mismos, los que remedian esos déficits sin necesidad de una enseñanza formal, **siempre que se hallen inmersos en un medio estimulante.** Para ello cuentan con dos poderosos medios de acción: su cuerpo y su pensamiento, que utilizan simultáneamente para el intercambio directo y funcional con el entorno.

Como producto de este intercambio, y favorecidos por la inserción en un medio social entretejido por un sistema de signos ya construídos (la lengua), los seres humanos -en su evolución ontogenética- plasman y modifican su pensamiento. Pero al mismo tiempo, y gracias a los intercambios adaptativos que establecen con el mundo que los rodea, se da la adquisición del lenguaje con la consecuente capacidad creciente de conceptualización.

De ahí que, el manejo del lenguaje (como signos compartidos para el intercambio de información) le permitirá al ser humano el desarrollo de las inagotables posibilidades de manipulación de la realidad, a fin de lograr un dominio claro del mundo como consecuencia de la aplicación de sus conocimientos. Es por ello que dice Piaget: "PENSAR, para el niño, SIGNIFICA MANEJAR PALABRAS". (2)

En el contexto de estos conceptos, Seymour Papert rescata la idea del maestro que considera su contribución más importante: **LA TEORIA DEL APRENDIZAJE, la cual no separa el estudio de cómo se aprende cualquier conocimiento del conocimiento mismo** (o, en palabras de Papert: "una teoría que no divorcia el estudio de cómo se aprende la matemática del estudio de la matemática misma." (3)

Papert se pregunta a continuación por qué estos aspectos epistemológicos del pensamiento piagetiano han sido descuidados

y, aún, desconocidos por muchos pedagogos. "Porque ... no ofrecían posibilidades de acción en el mundo de la educación tradicional" (4), puesto que, dicho método de educación adopta un conjunto estructurado de conocimientos que partiendo de la teoría, o de reglas abstractas y, por ende, generales, va a lo particular y concreto, a la cotidianeidad de las cosas que rodean al ser humano. En cambio, el conocimiento acumulado del hombre -tanto filo como ontogenéticamente- recorrió el camino inverso: primero se descubrieron, o destacaron del entorno, cosas o hechos concretos y particulares y, poco a poco, se fue afinando y elaborando en él la capacidad para referirse a patrones abstractos, o teorías explicatorias de la realidad, que subyacen a múltiples fenómenos cotidianos. Gracias a estos modelos explicatorios, le fue permitido al hombre aumentar su eficiencia y su dominio manipulatorio del entorno.

En contraposición, al partir la enseñanza tradicional de lo abstracto para ir a lo concreto (metodología deductiva), se transformó en una enseñanza "bancaria", cuya única preocupación era averiguar y mejorar los métodos que hagan posible "hacer entrar" en la cabeza de los alumnos la mayor cantidad de conocimientos posibles.

Desde la década del 60, la computadora como extensión del sistema neural humano (que permite potenciar las posibilidades de manipular y transformar la realidad), más los avances en la informática con las propuestas de la IA de emular con el computador el comportamiento-comunicativo-inteligente del hombre, posibilitaron a Papert plantearse un objetivo ambicioso: recrear las ideas de Piaget dentro de un contexto actualizado y utilizando los adelantos tecnológicos para romper con el planteo esquizofrenizante en la adquisición de la ciencia. Es decir, se trata de que la ontogénesis en la adquisición del conocimiento acumulado coincida con la forma en que el hombre aprehende la realidad, partiendo de lo concreto para llegar a la abstracción y construyendo a la par -como dice Piaget-, sus propias estructuras mentales para adaptarse a la realidad siempre cambiante del medio.

De esta forma es como nace LOGO en el equipo de investigación del MIT, donde las ideas piagetianas son colocadas en un marco teórico diferente tomado de un área de avanzada de la informática: la Inteligencia Artificial.

Ahora bien, nosotros sabemos que, en sentido estricto, la IA se ocupa de ampliar la capacidad de las computadoras para realizar funciones que se considerarían inteligentes si las realizaran los seres humanos; como tal, sería una rama de la ingeniería avanzada. Pero para lograr esta meta ambiciosa, los proyectos de IA necesitan abrevar en otras disciplinas: la medicina, la psicología, la lingüística, que le brindan aportes acerca de la naturaleza profunda de los mecanismos del aprendizaje y de la comprensión. Así nace una nueva disciplina que engloba todo esto: la Psicología Cognitiva, cuyo objetivo es brindar el marco teórico y la metodología de investigación para trasvasar estos

conocimientos a teorías computacionales de frontera. Así, la IA puede dar forma concreta a ideas sobre el pensamiento que antes pudieron parecer metafísicas. (Un ejemplo de ello, sería la concepción de la Escolástica de la existencia del alma basada en la supuesta inmaterialidad del "concepto").

En tanto los psicólogos cognitivos utilizan las ideas que van surgiendo de la IA para construir teorías científicas formales sobre los procesos mentales, los niños y jóvenes que utilizan estas nuevas metodologías surgidas de la IA usan las mismas ideas de manera espontánea y personal para **pensar sobre sí mismos** y para **construir sus propias estructuras mentales.**

Para clarificar lo antedicho, no olvidemos que el emisor de un mensaje comunica al receptor -junto con el mensaje- sus propias estructuras mentales. Es decir, en la medida que el lenguaje es comunicado por medio de una cadena de palabras lineal y relativamente lenta, ello permite construir al receptor la estructura mental apropiada para la recepción exitosa del mensaje. Esto mismo es lo que se trata de trasvasar a la interacción con el computador inteligente, tratando de emular el comportamiento comunicativo del hombre.

Para ello -y porque el significado de una oración o frase es, con frecuencia, más que la suma de significados de sus partes-, es necesario detallar al extremo los pasos que lleva implícita cualquier idea que queramos definir, debiéndose, asimismo, marcar las diferencias producto de los distintos contextos en que está inserto un vocablo determinado.

En la medida en que se debe realizar este proceso -que implica analizar detalladamente los distintos pasos implícitos en nuestros procesos mentales en función de la comunicación del pensamiento-, se piensa sobre uno mismo y se hacen conscientes nuestras propias estructuras mentales. Si no lo lográramos sería imposible alimentar al computador.

Esta es la base de la teoría del aprendizaje inserta en LOGO: la CONSTRUCCION consciente de nuestras formas de aprehensión y comunicación del conocimiento, como asimismo de las capacidades operativas basadas en el manejo de símbolos, con la consecuente aceptación y respeto por las diferencias individuales de de acercamiento a la realidad.

* LOGO COMO LENGUAJE DE COMUNICACION:

En los procesos de aprendizaje es muy importante la función de la comunicación de una persona a otra. Sin ella -si hay interferencias o incomprensión-, el aprendizaje falla y no se da. De ahí, la importancia del lenguaje.

Pero nosotros sabemos que el lenguaje va mucho más allá de las simples palabras. Obedece a una serie de sutiles convenciociones sociales, donde es tan importante COMO se dice algo tanto

como LO QUE se dice. Es decir, para comprender y captar un mensaje en su exacta dimensión debemos tener en cuenta tanto el lenguaje literal como el metalenguaje que lleva implícito.

Cuando hablamos de metalenguaje, hacemos referencia tanto a la acepción dada en la lógica -sistema de símbolos que hace referencia a otro sistema de símbolos en niveles jerárquicos de abstracción-, como al enfoque que se le otorga en la psicología: el sentido oculto, la lectura entre líneas o el mensaje subyacente que nos habla de la intencionalidad del emisor, puesto que, desde este ángulo, muy a menudo el significado transmitido es completamente diferente de la interpretación literal de las palabras.

Es por ello que, analizar lo que se dice no es una tarea simple, sino que implica poner en marcha un conjunto complejo de **mecanismos procesadores** de la información.

Veamos cómo se da ello: El discurso humano supone no sólo la intencionalidad cargada de sentido del emisor, sino también las características de los interlocutores que posibilitan o no la captación del mismo: conocimientos, entorno socio-cultural, razones que nos llevan a tomar parte en la comunicación establecida, etc. Por ello debemos tener en cuenta que, aunque en teoría toda persona alfabetizada puede leer lo escrito en un libro cualesquiera, sin embargo esto no es así. Cuando hablamos de "leer" no sólo nos referimos a la recorrida visual sobre las palabras, sino que hacemos referencia a la aprehensión y comprensión de la conceptualización implicada por el contexto y discernible en base a la masa de conocimientos preexistentes del receptor, como asimismo, al metalenguaje subyacente en el texto. Por ejemplo, si le diéramos a leer este trabajo a un niño de 12 años no sería capaz de leerlo, aunque tenga acumulada una experiencia lingüística de 10 años de evolución. (Recordemos aquí que la instauración del lenguaje se posibilita alrededor de los 2 años de vida del ser humano). Lo mismo pasaría si se lo diéramos para su lectura a una persona poco culturalizada, cuya capacidad de aprehensión se hallase a niveles muy concretizados, con un vocabulario escaso y muchas veces ambiguo y con estructuras muy simples de pensamiento.

Por qué se hallarían incapacitados para leerlo? Por lo siguiente:

1) El lenguaje, cuyo objetivo es el comunicar información de una persona a otra tiene que haber sido individual y previamente internalizado mediante el aprendizaje lingüístico y su conceptualización consecuente. Sabemos también, que el aprendizaje del lenguaje es un proceso que atraviesa diferentes etapas: de la palabra-rótulo y de un lenguaje autístico de los primeros años de vida va evolucionando al pensamiento inteligente-comunicable y, por ende, "dirigido a..." propio del lenguaje socializado, que supone a un interlocutor o "un otro" con quien interactuar.

2) En la mente humana, la información se representa -como producto de esa internalización- mediante una **red interconectada** e intrincada de construcciones conceptuales.

3) En la transmisión de la información -tanto verbal como escrita- el emisor apunta a comunicar, a sus eventuales interlocutores y/o lectores, no sólo una cantidad de conceptos, sino también las **estructuras mentales** organizativas de sus construcciones conceptuales. Pero para que esto sea posible, el sujeto receptor del mensaje debe "sintonizar la misma longitud de onda"; es decir, debe hallarse en un nivel de abstracción y conceptualización semejante, a fin de captar y comprender tanto "lo que se dice" como elmetalenguaje que se desprende del "cómo" sintáctico del discurso. En otras palabras,

4) **las estructuras mentales son complejas redes interconectadas y multidimensionales, mientras que el lenguaje se comunica mediante una cadena lineal de palabras sucesivas.** Estas palabras -verbales o escritas y de una en una- tienen que permitir que el receptor, de alguna manera, construya una estructura mental apropiada a la emisión originaria.

Son por estas razones que el lenguaje transporta gran parte del significado a través de convenciones sociales que nos entretejen a todos y, al mismo tiempo también, el lenguaje se apoya en la poderosa maquinaria procesadora que el interlocutor debe poner en marcha para **analizar, interpretar y elaborar** la información que recibe (y que el hablante espera que el oyente ponga). Es por ello que **siempre se comunica más de lo que se dice explícitamente.** Así ambos seres interactuantes -emisor-receptor- se enriquecen como resultado del intercambio, o, por el contrario, al no haber llegado al mismo nivel de conceptualización, la información muestra"interferencias" que el emisor debe subsanar.

Un ejemplo claro de esto es lo que sucede en la educación tradicional. Se parte de la concepción de la enseñanza con una metodología "bancaria": el alumno es una inversión a largo plazo en quien se debe depositar la mayor cantidad de conocimientos posibles -en forma dogmática, en la mayoría de los casos- (enseñanza enciclopedista), venciendo las más de las veces las resistencias que opone al conocimiento (incomprensión, dificultades de de captación que se solucionan con la memorización a ultranza, etc.) El conocimiento, de esta manera es abstracto y abstruso -cuando no totalmente críptico- accesible sólo para una élite de "iniciados" o "más brillantes" porque tienen un background intelectual hogareño que les hace familiar lo que escuchan o que los ayuda a decodificar el mensaje. Ni qué decir, si para el mismo maestro lo que transmite es, asimismo, ininteligible, y repite "como loro" un discurso que también le es ajeno e incomprensible, o poco claro.

En este contexto tradicional el puntaje cobra un interés primordial. Es una forma de premio o castigo que se dispensa al aprendizaje memorístico, donde se coarta el ingenio y el razona-

miento porque se debe repetir el "discurso instituído y aceptado" sin importar si ha sido comprendido y aprehendido, donde para resolver un problema se deben seguir los pasos que "alguien imaginó antes" y donde se castiga, o no se admite, el acceso a la solución por otra vía distinta a la propuesta por la cátedra, etc. Ante esta situación autoritaria y opresiva -ya que el poder lo tiene el docente porque tiene "el conocimiento" (?)- los alumnos reaccionan con rebeldía, pasividad, desgana, desatención, "cerrándose" o bloquéandose a cualquier acercamiento y, por ende, al conocimiento que vivencian como la fuente de sus conflictos.

La teoría del aprendizaje que subyace en LOGO, de ser cada uno constructor de sus propios modelos de pensamiento ayudado por la tecnología y orientado por el docente -como fuente estimuladora y de consulta-, permite una comunicación armónica.

El alumno deja de ser el "objeto" que se debe educar, moldear y ser receptáculo de conocimientos, para **transformarse en sujeto y artífice creativo del aprendizaje**, gracias a:

* que la **actividad está centrada en el alumno**, en vez de estar centrada en el docente -propia de los métodos tradicionales-.
* la apropiación del conocimiento: hacerlo propio mediante el accionar sobre un "microcosmos material obediente", a partir del cual los usuarios pueden **construir e interiorizar**, verbalizándolos, un conjunto de pasos que reflejan el universo de su pensamiento y que pueden cotejar en esa situación simbólica que simula la realidad.
* una enseñanza individualizada; es decir, aquella manera de enseñar que empieza por considerar el hecho de que todos los seres humanos difieren entre sí y presentan características que les son propias: nivel de inteligencia, manifestación de su temperamento y carácter, tiempos internos distintos, formas operacionales diferentes, etc. Dentro de este marco, cada participante recibe -en función de sus necesidades y sus aptitudes- un programa de estudio constituído por datos que debe asimilar, y por trabajos y ejercicios que efectuará -sea sólo, sea en grupo-, estimulado e incentivado por sus logros y, más aún,por sus equivocaciones: que lo llevan a buscar otros caminos y otras soluciones más acertadas.
* **se da el progreso de acuerdo al propio ritmo de cada alumno.**
* la necesidad de **dar mensajes unívocos al computador**, lo ayuda a **clarificar su pensamiento y a analizarlo en un gran nivel de detallismo.** Hacer que una computadora entienda un lenguaje natural no es simplemente que comprenda sustantivos y verbos, es mucho más. La comprensión de textos y oraciones requiere un complejo nivel de detállismo. El usuario debe tener claro que, en una simple aserción humana, hay una gran cantidad de conocimientos que no se hallan explícitos, pero que están implícitos en la misma, y que forman parte de nuestra experiencia cultural y de nuestras costumbres.
* el poder ensayar modelos de pensamiento o descubrir nuevos modelos en forma **experimental**, donde el usuario puede realizar o solucionar el problema propuesto, imponiendo a la computadora su ejecución paso a paso y en forma **modular** (dividiendo un

problema en partes para poderlo manipular mejor), con la posibilidad de modificar los pasos o el programa de instrucciones de manera fácil, y, al mismo tiempo, con la posibilidad de hacer de un conjunto de instrucciones **un todo.**

* que no existe el **concepto de error** en el sentido de los otros lenguajes de programación o de la educación tradicional. Ante el error, el sistema le solicita precisiones; por consiguiente, **no es culpógeno.** De esta manera, **la falta de éxito es percibida como una etapa hacia una elaboración más completa de procedimientos** que están más en conformidad con los pensamientos del usuario. Permite, además, una **mejor comprensión de los errores:** al no ser un hecho frustrante, estimula y alienta la investigación de nuevos caminos o nuevas soluciones.
* **el poder de motivación del sistema:** puesto que, es una fuente de estímulos permanente. No se ha encontrado a alguien que abandonara un procedimiento antes de ponerlo a punto e, incluso, se ha visto venir a los niños y adolescentes a la vez siguiente con ideas nuevas para resolver problemas que a sus juicios habían resuelto mal.
* al hecho de ser una **auto-socio-construcción del saber y la destreza en un contexto heurístico.** Es decir, es "autoconstrucción" porque el usuario del sistema -por un camino que le es propio- se construye **su** saber, pero no construye **el** saber; y, al mismo tiempo, es "socioconstrucción" porque cada participante no construye su saber a solas: interviene de manera importante el docente "poniéndolos en situación" y evacuando dudas u orientando, y los demás usuarios, intercambiando experiencias, logros y dificultades en el seno de una **estructura horizontal.** En esta estructura horizontal, el docente es un acompañante del progreso en el conocimiento, un orientador (no el dueño del poder del conocimiento).

Ahora bien, el educando se transforma en sujeto y actor en la adquisición del conocimiento porque el lenguaje LOGO está constituído por:

› un conjunto de **palabras primitivas** que permiten cambiar el estado de la tortuga, modificando:
  + ya sea su **posición**, mediante los términos "adelante" y "atrás",
  + ya sea su **dirección:** "izquierda" y "derecha".

  Este conjunto de palabras primitivas traducen conceptos de base.

› a partir de las palabras primitivas, el usuario crea otras palabras: los **procedimientos** que necesita para resolver sus problemas.

De esta manera tan simple, y con el lenguaje del usuario, se elaboran los **"programas"** que consisten en un conjunto de palabras primitivas y de procedimientos que permiten manejar "objetos" (números, palabras, listas, listas de listas, matrices, etc.), y también, distinguir entre la noción de "continente" y "contenido". (El "continente" posee un nombre que es una palabra que le fue asignada por el usuario para nombrarlo, y el "conte-

nido" es un objeto definido con anterioridad por el mismo usuario y que es designado por el nombre de su continente. Si el contenido es una palabra, ésta puede a su vez ser el nombre de un continente. Por ejemplo,

```
para "cuadrado
repetir 4 (adelante 100 derecha 90)
```

donde "cuadrado" es el continente, y el "contenido" son las instrucciones para la construcción del cuadrado.)

En la medida que el lenguaje que se utiliza es, en su mayoría, producto de la creación del usuario que nombra y designa objetos y las distintas variables para la construcción del conocimiento y para la comunicación con la computadora, es un lenguaje de comunicación accesible e inteligente que permite un diálogo fluído con un "semejante" en la lengua materna.

Este lenguaje de IA permite:

* **NOMBRAR** todo objeto que el usuario necesite para resolver sus problemas.
* **DESCOMPONER:** la capacidad para aprender que c/u.tiene depende de su habilidad para generar una buena descripción de la tarea que se trata de dominar y manipular. Para ello es necesario descomponer el problema; es decir, particionar la realidad y distinguir los objetos, o lo que es lo mismo, dividir dicho problema en módulos de fácil acceso.(Por ejemplo: para construir una "casa" descompongo el problema en módulos: techo,frente, chimenea, puerta, ventana, etc.)
* **CONSTRUIR:** a partir de las palabras primitivas y de los procedimientos se pueden construir "ladrillos"con los cuales, y a partir de los mismos, es factible construir nuevos procedimientos y armar estructuras más complejas.
* **GENERALIZAR O ABSTRAER:** El primer paso relacionado con la noción de generalización consiste en la posibilidad de elegir los **parámetros** del procedimiento. Por ej:hacer una puerta rectangular consiste en darle "altura" y "ancho" que son dos parámetros dimensionales. Estos parámetros o variables -en la medida que los defino- permitirán construir y generar puertas de todas las dimensiones:

```
para "rectángulo :altura :ancho
repetir 2 (adelante :altura
          derecha 90
          adelante :ancho
          derecha 90)
```

Cuando lo quiera ejecutar invocaremos lo siguiente:

```
rectángulo 100 50
```

donde 100 y 50 serán los parámetros que serán asignados a las variables "altura" y "ancho".

Y, el segundo paso dentro de la generalización, como síntesis o como un mayor nivel de abstracción, es la

recursión: el arte de empezar de nuevo el mismo procedimiento con parámetros o contextos distintos, en el interior mismo del procedimiento reiniciado.

* HACER RAZONAR estableciendo un nexo entre los objetos.

* LOGO COMO UN ENTORNO O AMBIENTE GRATIFICANTE.

Si partimos de la base de que en la construcción y generación del pensamiento es tan importante lo cognoscitivo como lo afectivo -dos aspectos esenciales y estrechamente interdependientes en toda conducta humana-, el ambiente en que se da esta elaboración constructiva de los procesos mentales cobra una importancia capital.

En la medida en que el ser humano percibe que puede ir comprendiendo la realidad e instrumentando su motivación y curiosidad para el accionar sobre el mundo, se autovalora y acrecienta su seguridad personal.

La afectividad (el "sentirse afectado por..." generando atracción o rechazo) **se transforma en el motor de la acción;** es decir, en la medida en que los distintos elementos o situaciones del medio despiertan nuestro interés nos ocupamos de ellos. De esta manera, al sentirnos atraídos por algo que despierta nuestra curiosidad, los sentimientos asignan un objetivo a la conducta, mientras que la inteligencia proporciona los medios, la "técnica" o el método para la aprehensión cognoscitiva.

Es por ello, que la vida afectiva y la cognoscitiva, aunque distintas, son inseparables e irreductibles una a la otra. El desarrollo de la inteligencia implica que haya intereses y curiosidades en el sujeto, y si el medio es rico en incitaciones, es estimulante -como decía Piaget-, se tendrá un desarrollo más avanzado.

El ambiente LOGO, como entorno estimulante, favorece la curiosidad, la capacidad de asombro y la creatividad humana al afinar y enriquecer cada vez más esos instrumentos de asimilación, los cuales permiten conductas adaptativas progresivas y cada vez más complejas frente a una realidad siempre móvil y cambiante, a distancias espacio-temporales crecientes tanto pasadas como futuras (recuerdos, proyectos, hipótesis, etc.).

Por otra parte, en la medida en que se respeta la individualidad de cada alumno y se estimula su creatividad, en un microcosmos experimental no coercitivo -que obliga al individuo a proyectarse al exterior, a imaginar y a tratar la información como un fenómeno de acción inmedianta sobre el ambiente-, **la educación se convierte en el proceso de totalización del yo** a partir de las múltiples interacciones con el entorno, permitiendo al educando adquirir dominio sobre sí mismo y sobre el mundo exterior.

En esa misma medida, **EL SER HUMANO APRENDE A NO TENER MIEDO A APRENDER.**

III.- L A E X P E R I E N C I A L O G O.

a) PRIMERA ETAPA:

Los dos primeros cursos de LOGO se realizan durante el año 1985, contando ambos grupos con 6 alumnos cada uno. Las edades de los mismos oscilan entre los 11 y los 18 años, siendo la edad promedio del curso: 14 años.

Los docentes son alumnos avanzados de la carrera de Ciencias de la Computación del Instituto, a los que la docencia asistida por computadora les resulta una experiencia fascinante por las múltiples posibilidades que brinda para el aprendizaje. Sin embargo, al comienzo de esta investigación no tenían experiencia como docentes, lo que los indujo a seguir las pautas aplicadas por el Ingeniero Reggini en la enseñanza de LOGO -dado el acopio experiencial y experimental que este investigador argentino había realizado en la materia-. Es así que se prefiere, puesto que no se sabe cuál puede ser la respuesta de los alumnos, trabajar sobre un material que ya estuviera hecho y probado eficazmente, siguiéndose por ello los lineamientos que Reggini plantea en su libro "Alas para la Mente".

Se asignan, entonces, funciones diferenciadas: uno, es el docente a cargo de la cátedra, y el otro, se desempeña como "asistente" y observador de la tarea del grupo. De esta manera consignan, día tras día, un informe pormenorizado de lo que va sucediendo (progresos, dificultades, trabajos realizados, etc.) para permitir el seguimiento y evaluación de la experiencia dentro del equipo de investigación.

Por los motivos antes mencionados, la orientación general del curso se dirige a la realización gráfica, que permite un umbral de comunicación muy cercano al cotidiano, donde el atractivo de la imagen se impone (para la mayoría de las personas sin distincion de edades) y les permite disfrutar de la verificación visual de los esquemas diseñados por la tortuga siguiendo el reflejo de sus pensamientos exteriorizados en órdenes. Se da, por consiguiente, importancia a las instrucciones de tipo gráfico y a las reglas sintácticas y ortográficas a fin de lograr que la tortuga obedezca.

Los alumnos, por su parte, descubren la necesidad de descomponer un problema planteado en una serie de módulos que facilitan la resolución del mismo, y que se pueden guardar para utilizarlos cuando hagan falta. Trabajando con los procedimientos modulares (o procedimientos de detalle) descubren fácilmente los inevitables errores que se cometen al realizar programas extensos. Se dan cuenta, también, de que el particionar la realidad simulada facilita probar, depurar y modificar esos módulos en forma independiente hasta lograr su optimización para recién incluirlos dentro del programa principal. Es decir, descubren que se pueden explorar ideas complicadas a partir de elaboraciones jerárquicas sucesivas.

Esto da cuenta de una teoría elaborada por Papert y Marvin Minsky sobre la modularización de las estructuras mentales en su adaptación a la realidad. Dicha teoría responde a la idea general de que todo sistema funciona sobre la base de subsistemas relativamente independientes entre sí. Ya Piaget consideraba que el pensamiento respondería también a este principio, puesto que la conceptualización se elaboraría en forma modular en una yuxtaposición de pequeñas entidades de conocimientos; es decir, más que un inmenso procedimiento de gran complejidad, sería un conjunto de numerosísimos procedimientos simples que permitirían el trasvasamiento de esa estructura a situaciones similares y su corrección o ajuste ante el hecho puntual.

En esta depuración de los errores o de los procedimientos que no funcionan bien, y al recibir el mensaje de mayor precisión, los alumnos deben descubrir la falla e interpretar la causa de la misma, reflexionando sobre cómo construyeron su planteo (reflexionan sobre sus pensamientos), y experimentan que -al equivocarse, sin reprimendas y sin coerción culpógena- las equivocaciones les permiten acercarse paso a paso a la meta propuesta y se transforman en una fuente de entendimiento. Así, poco a poco, se van planteando nuevos problemas de dificultad creciente.

Pero, una característica que se evidencia en estos dos primeros grupos es la dependencia que muestran respecto al trabajo. Se mueven como en la escuela, esperando que el profesor les sugiera ideas o ejercitaciones posibles. Ante las propuestas, que se iban realizando en su mayoría en forma pareja por los distintos integrantes de los cursos, se van viendo las preferencias de los alumnos y en función de las mismas los docentes les van brindando las ideas o proyectos posibles para que realicen. Es decir, la función del docente es bastante paternalista y existe menor flexibilidad en los conocimientos impartidos; la experimentación está como predeterminada de antemano.

Es así que lo que se percibe es que, en parte, por la preeminencia del color y por la orientación del curso hacia las instrucciones de tipo gráfico, los alumnos tratan de emular con el computador los juegos electrónicos. Se preocupan sobremanera por la forma y el color, y el impacto que puede causar una buena presentación visual para el que se coloca ante la pantalla. Los juegos elaborados y construídos por los alumnos (match de box, dados, ruleta, carrera de caballos, tateti, etc.) impactan por el colorido y la conformación, pero no hay una real preocupación por establecer reglas lógicas que validen y controlen los distintos pasos. Son juegos planteados para jugar dos operadores entre sí utilizando el computador como una herramienta televisiva. Son juegos ingeniosos que utilizan variables y vectores, la elección de números al azar por la aplicación de la teoría de la probabilidad en la elección de números o movimientos, unida a procedimientos simples del tipo "si...entonces..." (ej, si el número es mayor que 4, entonces ir hacia adelante), o el planteo recursivo aplicado al grafismo donde el procedimiento se llama a sí mismo, pero donde la pantalla se reduce a ser -en la mayoría de los ca-

sos- una mera planilla de dibujo. Un ejemplo de ello es el tateti: a través de distintos procedimientos simples el computador dibuja el tablero y pregunta a los oponentes dónde quieren colocar sus fichas; cada participante le da una ubicación en la matriz y la computadora dibuja, pero no controla si el espacio está ocupado o no por una ficha anterior.

Un solo trabajo escapa a la graficación. Es el realizado por Florencia de 14 años, que trabaja con listas de palabras conformando una pequeña base de datos que permite a la computadora, al random, construir oraciones con este esquema:

nombre propio masculino - conjunción - nombre propio femenino- - verbo - complemento directo - complemento de lugar.

La situación surge porque Florencia, por su despertar adolescente y, porque hay un compañero del grupo que la atrae, quiere que la computadora, al azar, la una -en ese "microcosmos simulado"- al chico por el que suspira.

Por otra parte, en lo que hace al ambiente de trabajo, los alumnos compiten sanamente entre sí para ver quién presenta la mejor realización graficada y se aportan datos o descubrimientos hechos en la implementación de sus procedimientos. Sin embargo, si bien existe colaboración en la comunicación de sus implementaciones, el resultado de la tarea es un resultado indivicual; no se logra un trabajo en equipo (un trabajo modularizado que sea un desafío a la imaginación y que, por la envergadura del mismo, se vean comprometidos y motivados los integrantes del curso a realizarlo entre varios).

b) SEGUNDA ETAPA:

Habiendo evaluado la experiencia anterior, se encaran los dos cursos de 1986 con una nueva orientación puesto que ahora se sabe que siempre que se incentive al alumno y se lo motive adecuadamente, siempre que el desafío se plantee en forma interesante, no existen barreras infranqueables para la concreción de cualquier hipótesis de trabajo por dificultosa que parezca.

Se decide también, no remarcar tanto la graficación y los procedimientos relacionados con la misma, sino tratar de que esos razonamientos más o menos intuitivos que los alumnos poseen los vuelquen en un material no figurativo y más abstracto, donde la computadora -como extensión del pensamiento humano- sirva para implementar y materializar sus representaciones ideacionales en un alto nivel de abstracción. Por consiguiente, la orientación del curso va a apuntar a crear, manipular y recuperar información simbolizada de cualquier modelo simulado de la realidad. Se apunta, también, a hacer incapie en las estructuras lógicas que subyacen y sustentan los acontecimientos del entorno cotidiano, donde el conocimiento comunicado a la computadora cobre dinamismo al ser procesado y, automáticamente, produzca respuestas y nuevos planteos que se derivan de ese conocimiento.

Es decir, lo que se quiere conseguir es que la computadora -como un alter ego adaptativo sofisticado- sirva para ayudar a construir nuevas hipótesis de trabajo y para dialogar creativamente con ella.

En función de ello se decide acentuar el rol protagónico de los alumnos y que el aprendizaje surja como resultado del medio altamente estimulante donde los profesores acompañarán y orientarán las inquietudes que generen los educandos y los ayudarán a seleccionar los proyectos que los motiven de acuerdo a sus intereses y habilidades personales. Simultáneamente se los incentivará permanentemente a mejorar sus procedimientos para acercarlos, en forma paulatina, a la optimización de los mismos.

En base a ello, los objetivos planteados son:
- motivar la investigación para la resolución de proyectos específicos;
- llevar a la praxis la idea intuitiva de modularización de los procedimientos;
- adaptar los problemas sugeridos por el alumno, de tal manera de utilizar nuevos conceptos teóricos;
- inducir al alumno hacia la reflexión de su propio pensamiento, logrando así perfeccionar su mecanismo deductivo, ya que la resolución de problemas es un aspecto fundamental en la educación;
- fomentar el trabajo grupal, de tal manera de comunicar diferentes ideas ante un mismo problema;
- lograr que el error se transforme en una pauta constructiva y no en un factor frustrante;
- convertir al docente en un integrante activo del grupo de aprendizaje, ya que por su conocimiento más profundo puede guiar la solución de problemas;
- desmitificar la relación alumno-profesor de manera tal de lograr un ambiente cómodo de trabajo y de plena confianza.

Una vez que aprendieron las "palabras primitivas" y las instrucciones más simples de LOGO, se hace incapie en las pantallas de texto -más que en las gráficas- para la comprobación y verificación, y en los mensajes o las aclaraciones que Logo solicita.

Al no apuntar tanto a la graficación, los alumnos se sumergen con sumo interés en:
- los engramas lógicos que subyacen y controlan los fenómenos del universo (si verdadero..., si falso...);
- en enseñar a "dialogar" a una computadora simulando un ser inteligente;
- a competir con la computadora como otro "yo" a quien se enseñó a "pensar" -explorando, por ende, cómo piensan ellos mismos- y escribiendo los procedimientos adecuados y los controles necesarios en listas cada vez más complejas que utilizan la recursión a niveles de abstracción mayores o el procedimiento al azar validado por parámetros lógico-matemáticos;
- clasificar palabras en categorías porque se necesitan para los ejercicios de diálogo, con lo que refrescan sus conocimientos

gramáticos y comprenden la idea general de que las palabras -como las cosas- pueden clasificarse en distintos grupos o conjuntos, y que hacerlo puede resultar útil, etc.

Todos los procedimientos se dieron en función de juegos y desafíos planteados por los docentes, que los alumnos enriquecían y optimizaban, cada vez más asombrados al ver trabajar a la computadora como imagen especular de sus estructuras mentales. Los trabajos realizados evidencian lo dicho: 1) el ahorcado: utilizando la pantalla gráfica y los textos, manejando una estructura de datos para analizar palabras por descomposición unitaria de caracteres (strings) con empleo de ciclos de control; 2) los dados; 3) el laberinto donde, basándose en funciones de tipo booleanas (salidas por verdadero o falso), mediante la simulación de sensores táctiles, se logra desarrollar un algoritmo que recorre un laberinto conservando siempre una pared a su derecha y cuando se encuentra con una pared aislada -tipo islote- se incorpora la función de "giro total alrededor de una figura", con lo cual se elabora un modelo lógico que responde a cualquier tipo de estructura física real, abstrayéndose de circunstancias particulares; 4) simular un juego electrónico donde una tortuga dispara sobre un blanco móvil, llevando a la práctica el uso de coordenadas para la validación y control de la posición de la tortuga en un instante, utilizando además el concepto geométrico de punto contenido en una recta, verificando la certeza de los disparos y el empleo de una rutina de números aleatorios para simular en la tortuga el movimiento libre; 5) desarrollo de un procedimiento que generara poesías al azar en forma totalmente aleatoria utilizando reglas sintácticas y un diccionario externo compuesto de sustantivos, adjetivos, verbos, adverbios, artículos; 6) el diálogo de un médico con su paciente, etc.

Esto es posible porque la edad promedio de los cursos es mayor (16 años) y porque se abandonan los trabajos prácticos fijos, generando un ambiente más propenso para la investigación, y totalmente libre en cuanto a la elección de los temas o programas a realizar. Además, en la medida que los trabajos a encarar son de mayor envergadura y con un alto nivel de complejidad, se produce el trabajo en pequeños equipos y con aportes esporádicos de los otros alumnos del curso.

Aquí se manifiesta claramente que los alumnos no tienen miedo de afrontar tareas o resolver problemas, por complejos que sean, en la medida que les resulte interesante la propuesta, y cuando el medio es estimulante y no coercitivo.

c) GEOMETRIA CONSTRUCTIVA:

Esta tercera experiencia es desarrollada por un alumno del Instituto como implementación de ideas contenidas en el libro "Turtle Geometry" de Abelson y diSessa. (5)

Debemos considerar dos ideas interrelacionadas que presiden el trabajo:

a) El computador es una herramienta para el aprendizaje de contenidos eidéticos diversos. El conocimiento del com-

putador en sí mismo, su arquitectura y su programación, no tiene por qué ser prioritario cuando se coloca la máquina en el ambiente escolar. Esta idea -implícita en Abelson y diSessa- es asumida y explicitada por el diseñador del curso. De esta manera, presenta al educando instrucciones de alto nivel desarrolladas por él mismo, que permiten minimizar el tiempo dedicado al conocimiento del computador y "saltar" más rápido a su uso como "pizarra electrónica".

b) La enseñanza de la matemática es más efectiva si su punto de partida, su escenario inicial, está constituído por situaciones comunes y concretas cuyos componentes pueden manipularse y colocarse en relaciones diversas uno respecto a otro. Este tipo de tarea facilita el señalamiento por el docente, o la advertencia espontánea por el alumno, de patrones primarios, en una primera etapa, y de patrones de patrones en etapas más avanzadas. De esta manera, la formulación abstracta de la matemática recupera a nivel individual el desarrollo histórico de esta ciencia, su "filogénesis". El niño o el adolescente fabrica sus abstracciones matemáticas y puede, así, gustar de ellas y usarlas con propiedad. El camino contrario -muy común y dolorosamente recorrido por casi todos nosotros- es la "receta" algorítmica dogmática ("menos por menos es más". Por qué? Algún maestro fue capaz de explicárselo, si es que usted se animó a preguntar?) y la abstracción, que debe estudiarse en primer lugar, para ver luego -si es que se llega a ver- cuál podría ser su "aplicación".

Por sus características, este curso está destinado a adolescentes a partir del segundo o tercer año de su escuela secundaria.

En cuanto a la metodología en general, primero se hace experimentar y formular hipótesis al alumno para luego sugerirle la teoría correspondiente. Cada unidad del curso se complementa con ejercicios variados para ser resueltos por los alumnos y, si es preciso, con la guía y ayuda del profesor.

La primera parte del curso familiariza a los alumnos con las herramientas de TLC-LOGO que utilizan durante el mismo: comandos, estructuras básicas de procedimientos, etc. Luego, se realizan los primeros procedimientos simples que denotan una geometría dinámica y variable. Investigando sobre estos procedimientos se llega a conclusiones sobre temas tan variados como trigonometría o constantes topológicas (referencias a características comunes de formas de objetos que permiten su clasificación: por ejemplo, cuadrado, triángulo, hexágono, etc. y sus procedimientos únicos de construcción semejantes al del círculo, ya que todas son "figuras cerradas" donde lo que varía son la cantidad de lados y el ángulo de giro) y curvatura intrínseca (la curvatura de una arco respecto a sí mismo, donde se relaciona la curvatura con el radio de una circunferencia; es decir, sin parámetros de referencia externos como podrían ser los ejes de coordenadas. La fórmula de la curvatura intrínseca es: curvatura = giro / avance).

Si bien, las constantes topológicas y las nociones de curvaturaa intrínseca son usualmente reservadas para cursos de nivel universitario, se pueden explicar perfectamente con esta metodología constructiva y son eminentemente claros para los alumnos, puesto que ellos al experimentar ven que el procedimiento es el mismo:

```
para "poly :lado :ángulo
        repetir siempre (adelante :lado
        derecha :ángulo)
```

Es decir, esta primera parte introduce a los métodos de la geometría constructiva o "geometría de la tortuga", y enseña a estimar las figuras geométricas no como entidades abstractas, sino como resultados de procedimientos computacionales simples que controlan a la tortuga.

En la segunda parte se aplican estos métodos simulando modelos de comportamiento animal y de crecimiento, que sirven de introducción a la "biología matemática". En esta segunda parte, se da énfasis a la investigación mediante la creación de grupos de trabajo que se dedicarán -según las preferencias de los alumnos- a temas biológicos (biología matemática), físicos, estadísticos, etc., bajo la supervisión y guía del profesor. (Por ejemplo, algunos de los temas propuestos versan sobre modelos de comportamiento animal -ortoKinesis y clioKinesis-, intensidad de la luz, probabilidades, sistemas numéricos, diseño artístico, diseño recursivo, etc.)

En la experiencia piloto realizada, se advierte que:

- la realización gráfica permite un nivel de comunicación muy cercano al cotidiano, y el atractivo de la imagen se impone a los alumnos y les permite verificar permanentemente sus logros y efectuar comparaciones para elaborar y construir sus propias reflexiones;
- al reflexionar sobre cómo construyen sus planteos o problemas a resolver, reflexionan sobre sus pensamientos y cómo encaran y manipulan cotidianamente el mundo que los rodea;
- experimentan que al equivocarse -en una situación no generadora de culpas-, las equivocaciones les permiten acercarse paso a paso a la meta propuesta y se transforman, de esta manera, en fuente de mayor comprensión y entendimiento de la realidad;
- los alumnos expresan entusiasmo y se hallan motivados para continuar el curso -cosa poco frecuente en otros cursos de matemática o geometría-;
- existe una perfecta comprensión del lenguaje: sin haber utilizado nunca previamente una computadora no evidencian dificultad para manejar el lenguaje LOGO;
- muestran una amplia captación de los contenidos, a pesar que uno de los alumnos integrantes de la experiencia tiene dificultades escolares en el aprendizaje de la matemática en su escolaridad habitual. Conceptos tales como curvatura intrínseca o nociones de topología no generan problemas, y son fácilmente construídas y aprehendidas a partir de la manipulación interactiva de ideas graficadas;
- es un estímulo permanente para la investigación por el carác-

ter dinámico que LOGO otorga a la geometría. Los alumnos exploran distintas posibilidades, y ello genera nuevos conceptos, incluso preguntas que el profesor no se había autoplanteado previamente y que lo obligan a investigar a él mismo -en conjunto con el alumno- para hallar la solución.

En cuanto a los aspectos negativos de la experiencia se ve: por un lado, poca participación de los educandos que han hecho carne el esquema tradicional de la educación y esperan -sobre todo en uno de los casos- en forma pasiva las instrucciones del profesor, con poca iniciativa propia, aunque se halla interesado en el curso. Por el otro, se evidencia la influencia de los video-games que los lleva a solicitar aprender -en ambos casos- a programar juegos similares a los comerciales. Es por ello, y en respuesta a estas inquietudes, que se trata de modificar la segunda parte del curso, proponiendo juegos que construirán los propios alumnos y que utilizan los conocimientos impartidos. Por ejemplo, utilizando lo aprendido en cómo se orientan los animales, enfrentar a dos tortugas con métodos de orientación diferentes para ver cuál es el más efectivo en diversas circunstancias, etc.

IV.- C O N C L U S I O N E S .

A través de la experiencia de investigación del TLC-LOGO y al seguimiento pormenorizado de los alumnos durante los cinco cursos realizados en el Instituto NCR de Ciencias de la Computación, durante más de un año, podemos arribar a las siguientes conclusiones:

* el uso de las computadoras, por sí mismas, no producen un beneficio educativo en forma automática. Para que su utilización contribuya efectivamente a ello, es necesario crear condiciones de aprendizaje en las que, utilizando los modernos adelantos informáticos, se genere un ambiente estimulante y no autoritario que favorezca la experimentación, verificación y creación de los alumnos. Para ello, la computadora no debe ser solamente un instrumento para preguntas y respuestas, de acopio de información, una mera calculadora o para realizar dibujos.

* el lenguaje utilizado -en este caso el TLC-LOGO- permite al educando comunicarse interactivamente en su lengua materna y crear sus propios procedimientos de acuerdo a sus características individuales.

* el método de aprendizaje al partir de la cotidianeidad de los fenómenos de la realidad para ir construyendo los modelos explicatorios y abstractos -la teoría-, posibilita la aprehensión y comprensión de las leyes lógicas que subyacen en el comportamiento de la realidad. De esta manera, el alumno entra espontáneamente en íntima relación con las ideas básicas de la ciencia y la construcción de modelos intelectuales que le per-

miten la apropiación y la interiorización del conocimiento.

* al permitir al alumno reflexionar sobre cómo piensa (pensar sobre su pensamiento) en su accionar sobre el mundo, se le posibilita exteriorizar creativa y conscientemente sus propias estructuras mentales para poder interaccionar unívocamente con el computador como un alter-ego inteligente, y desarrollar cada vez más sofisticadamente sus capacidades operativas basadas en el manejo de símbolos.

* permite descubrir que la forma de encarar la resolución de problemas es particionándolos en forma modular, construyendo procedimientos simples que posibilitan manipular la realidad con un alto nivel de abstracción, que permitan luego ser aplicados en procedimientos complejos para construir el todo. Este particionamiento de la realidad simulada, facilita probar, depurar y modificar esos módulos en forma independiente hasta lograr su optimización, que es como funcionan nuestras estructuras mentales en su adaptación a la realidad y que nos permiten la aplicación posterior de esas estructuras simples a situaciones similares y más complejas.

* el error -o la falta de éxito en la situación simulada- se percibe como una etapa hacia una elaboración más completa del conocimiento. Deja, así, de ser culpógeno y frustrante, para alentar la búsqueda de nuevas soluciones y se pierde el miedo al aprendizaje. Por otra parte, al poder observar inmediatamente las diferencias entre lo que desean hacer y lo que realmente sucede, les permite comprender dónde se halla la falla en su razonamiento.

* el ambiente estimulante y creativo, favorece la aceptación y el respeto por las diferencias individuales de acercamiento a la realidad, a la par que cobra conciencia de que en la resolución de problemas los caminos pueden ser variados. De esta manera, se rompe con el dogmatismo del conocimiento y se estimula la experimentación, sin dejar de ver aquí consecuencias cívicas: La pluralidad y la tolerancia no son solamente buenos consejos, sino vivencias rotundas. Es imposible que la enseñanza "sentenciosa" contribuya a desarrollar ciudadanos democráticos. La alternativa crucial es, en ese entorno autoritario, la verdad del maestro-führer o el castigo; por consiguiente, en el futuro, cada educando tratará de ser un "pequeño führer" en su contexto social porque no conoce otro modelo de comunicación y acción colectiva.

* se acrecienta la tarea de equipo y la colaboración de los participantes en la adquisición del conocimiento. El docente -en esta relación horizontal- es una guía orientadora y acompañante activo del progreso en el conocimiento, al mismo tiempo que aprende de los que aprenden y se replantea nuevas formas de acercamiento a la realidad.

* el progreso en el aprendizaje se da al ritmo propio de cada uno, y por ello resulta gratificante.

* los alumnos acrecientan su autoestima y su seguridad al ver lo que son capaces de hacer, y ello redunda en un mayor rendimiento y eficacia, no sólo en sus tareas escolares sino en su vida cotidiana.

* al ser el educando el centro y el actor de su propio aprendizaje, se rompe con la enseñanza como una estructura de poder donde el que sabe -el maestro- ejerce su poderío, dado por una burocracia educativa que lo autoriza a dictar "recetas" y algoritmos dogmáticamente, sin explicaciones, que en la mayoría de los casos ni siquiera se las podría dar a sí mismo.
  Recordemos, aquí, que la mayor parte de los debates actuales se centran alrededor de la "pedagogía del docente", quien espera de la informática un apoyo por el cual él pueda lograr que un alumno asimile un contenido temático que, con frecuencia, el docente no tiene cómo replantear, o que sirva como mero elemento de fijación del conocimiento a través de la repetición del tema -cosa que ciertos alumnos necesitan y por ello hacen perder el tiempo de todos, decreciendo el ritmo de la clase-. La informática, de esta manera, se transformaría en el elemento ideal de la pedagogía del refuerzo, en vez de ser el instrumento que permite construir los modos de pensamiento y revela, paso a paso, el avance intelectual del educando.

Por todo lo dicho anteriormente, la reflexión que nos cabe es la siguiente: Frente a una tecnología nueva, como es la informática, y con un lenguaje de IA como es el LOGO, el problema fundamental que debemos plantearnos es saber **quién** puede apropiarse de ellos y **con qué fin**. El docente, para reforzar un **poder** que siente cada vez más discutido? El propio educando para construir su propio modo de acercamiento a la realidad y sus propios modelos de la misma? Ambos?

Según sea la respuesta, así serán las generaciones futuras. La responsabilidad que toca a quienes determinan las políticas educativas es muy grande. Este trabajo quiere contribuir como un aporte al gran debate.

V.- B I B L I O G R A F I A .

(1) PIAGET, Jean : "La représentation du monde chez l'enfant"; P.U.F, París, 1.972, pág. 72.
(2) PAPERT, Seymour: "Desafío a la mente. Computadores y educación", edics. Galápago, Buenos Aires, 3a.edición, Mayo de 1.984, pág.181.
(3) PAPERT, Seymour: obra citada, pág. 182.
(4) PAPERT, Seymour: obra citada, pág. 182.
(5) ABELSON, Harold y diSESSA, Andrea: "Turtle Geometry", M.I.T. Press, Series in Artificial Intelligence, Cambridge, Massachusetts, 1.981.

BUENOS AIRES, Julio de 1.986.-

PAUTAS DE DISEÑO PARA REDES TELEMATICAS

Luis César Giménez
Néstor Carlos Galina
Bariloche - *Argentina*

INTRODUCCION:

Los grandes avances producidos en el campo de la tecnología microelectrónica durante la década de los años 70, han hecho posible obtener componentes tales como microprocesadores, memorias y otros circuitos VLSI a mucho menor costo, más confiables, veloces y en general con mejores y nuevas características funcionales.

Estos hechos han repercutido favorablemente en otras tecnologías basadas en la electrónica como lo son la informática y las telecomunicaciones.
La situación antes señalada constituye una de las principales causas que motivaron la difusión y uso generalizado de computadoras en todas las áreas de la actividad social, ya sean de producción, administración, enseñanza u otras.

El carácter abarcativo e invasivo de la herramienta informática, su amplia difusión tanto en lo que hace al tipo de aplicaciones como a su distribución geográfica, la necesidad de disponer en puntos distantes la capacidad de procesamiento de las computadoras, la conveniencia de compartir recursos como unidades de memoria de almacenamiento masivo e impresoras y las mejoras producidas en el terreno de las telecomunicaciones, son algunos de los principales factores que han llevado al surgimiento de amplias Redes Telemáticas.
Con este neologismo se busca explicitar la simbiosis, característica de los últimos años, en la cual se hallan involucradas dos de las tecnologías más significativas de nuestro tiempo, las Telecomunicaciones y la Informática.
Tal es su importancia que actividades como la bancaria, ventas de pasajes aéreos y otras, no son posibles de realizar actualmente sin dichas tecnologías.

A éstas, las Redes Telemáticas, las podemos considerar formadas por dos Subredes. Estas son:

1) Subred de Comunicaciones.

2) Subred de Proceso.

La Subred de Proceso es la que integran los recursos (terminales y computadoras) y que son de interés para el usuario. Estos recursos están conectados a los nodos de conmutación de la Subred de Comunicación, que juegan el papel de nexo de unión.

La Red de Comunicación que une a los usuarios puede estar organizada en conmutación de circuitos o de mensajes.

La diferencia entre ambas es la siguiente:

En una Red de Conmutación de circuitos, previo a la iniciación de la comunicación, se deberá crear un camino físico, el cual se mantendrá durante toda la transmisión.

En una Red de Conmutación de mensajes, éstos se van almacenando en los nodos intermedios de la ruta y esperan en cola hasta que puedan ser enviados al nodo próximo, al quedar libre la línea de salida. Esto es lo que comunmente se llama de almacenaje y envío (Store and Forward).

Si los mensajes se dividen en paquetes podremos hablar de conmutación de paquetes.

A su vez, éstos pueden ser de longitud variable hasta un máximo.

De esta manera, varios paquetes del mismo mensaje pueden estar siendo transmitidos simultáneamente, lo que resulta en una mayor rapidez en la transmisión.

En cambio, la conmutación de circuitos requiere tiempos de conexión mayores, y esto resulta preponderante cuando estamos en Redes muy activas, donde tambien resulta dificultoso establecer la conexión física de extremo a extremo.

Este inconveniente no se presenta en un sistema de conmutación de paquetes, pero aparece el problema de definir la magnitud de los retrasos nodales los cuales requieren que sean minimizados.

Retomando el objetivo fundamental de este trabajo, tenemos que considerar que uno de los problemas en el diseño de las Redes Telemáticas no reside en si existe tecnología disponible para el proyecto en cuestión, sino que el mismo se encuentra en como adoptar mejor la que existe, de modo de poder satisfacer las necesidades, generalmente cambiantes y retadoras, de las Organizaciones Comerciales y Gubernamentales.

Las Redes Telemáticas permiten la utilización más eficiente de las computadoras centrales, mejoran los controles cotidianos de una organización al proporcionar un flujo más rápido de la información, proporcionan servicios de conmutación de mensajes para que las terminales puedan comunicarse entre sí. Por lo general, ofrecen un intercambio de datos mejor y más oportuno entre sus usuarios y colocan la potencia de las computadoras a disposición de los usuarios.

Generalmente, los requerimientos de un sistema telemático surge de la necesidad de las administraciones de las Organizaciones que necesitan tener un panorama actualizado de las operaciones de sus filiales diseminadas geográficamente, las cuales, tienen como misión una mejor atención de sus clientes, vigilar de cerca las actividades críticas y por sobre todo hacer frente a la competencia.

Esta situación descripta, no sería factible realizarla sin contar con una recolección, procesamiento y distribución rápida de la información comercial.

A su vez, los avances en el diseño de computadoras y las significativas reducciones en el costo por operación junto a las ideas creativas en las aplicaciones de las computadoras, han aumentado el uso de los Sistemas Telemáticos o de Comunicación de Datos, para transmitir información entre ubicaciones comerciales ampliamente separadas y los recursos instalados en estas ubicaciones.

Esto hace que la administración pueda conocer al cabo de algunos segundos, cuál es la situación en alguna filial o sucursal, en el país o el mundo.

Resulta necesario también, que el diseñador de una Red Telemática, identifique la diferencia entre enfoque empírico (de prueba y error) respecto al enfoque de sistema en el cual se tratan de identificar las influencias y limitaciones sobre el resultado deseado del diseño, evaluándolos en términos de su impacto en los procesos Hardware, Software y recursos humanos.

ENFOQUE DE SISTEMA AL DISEÑO DE REDES TELEMATICAS.

Si bien existen varias maneras de realizar el mismo, podemos identificar 10 pasos básicos, los cuales son:

Paso 1) - Estudio de oportunidad.

Paso 2) - Estudio de profundidad, análisis y diseño.

Paso 3) - Recopilación de información entre las áreas afectadas.

Paso 4) - Estudio de interacción entre áreas afectadas.

Paso 5) - Evaluación y comprensión del sistema existente.

Paso 6) - Definir los requerimientos de la Red.

Paso 7) - Diseño de la Red Telemática propuesta.

Paso 8) - Desarrollo de las comparaciones de costo.

Paso 9) - Presentación de la Red.

Paso 10) - Implementación, seguimiento y revaluación del proyecto.

Podríamos agregar además los siguientes puntos:

- Documentación del Sistema.
- Control del proyecto.
- Pruebas y puesta a punto.

- Simulación.
- Formación.
- Arranque.

1) ESTUDIO DE OPORTUNIDAD:

El proyecto surge de una serie de ciertas ideas sobre los beneficios potenciales, normalmente de tipo financiero, que tal sistema podría aportar a determinada empresa, el objeto del estudio de oportunidad es el de investigar esta idea y producir una solución de principio en la que se resalten los beneficios y costos correspondientes, se tiene que llevar a la decisión sobre si es o no rentable financiar un estudio más detallado.

Se divide en dos partes:

a) Definición inicial: en esta se establece la viabilidad y necesidades del usuario para el proyecto (idea propuesta).

b) Definición detallada: se estudiará la rentabilidad del sistema preliminar y se considerarán todas las alternativas posibles. Además se trazarán planes para un estudio detallado en el que se indiquen los costos que implicaría un paso posterior.

Evidentemente, se requiere tener bien definido el problema a solucionar antes de buscar la solución al mismo, para lo cual resulta necesario disponer de una razonable cantidad de tiempo que permita lograr la definición más exhaustiva posible. Suele ocurrir que lo que se presenta como problema, solo es un síntoma del problema verdadero.

Deben ser coincidentes los objetivos de la Red Telemática con los propio objetivos de la Organización o Empresa.

Se deberá definir con claridad y analizar las ramificaciones del problema, señalando las interrelaciones entre el problema específico y cualquier otro problema, actividades o cosas descubiertas.

Deben ser expresados los objetivos a satisfacer al igual que el alcance del proyecto e indicar las áreas de la Organización incluídas o excluídas de la definición del problema.

Es fundamental que lo descripto anteriormente sea demostrado en forma clara y descriptiva, con una buena explicación de la metodología empleada para la solución de los problemas presentados.

Para esto, se deberán incluír tablas, gráficos, planos de distribución de equipos, mapas, y todo lo que pueda resultar útil para presentar con mayor claridad la problemática

2) ESTUDIO DE PROFUNDIDAD, ANALISIS Y DISEÑO:

En este paso, se deben preparar el plan de ataque y la estrategia a seguir para encarar la solución del problema, previamente definido.

Este paso resulta ser uno de los más importantes ya que en el mismo se deberán incluir las actividades necesarias para realizar los pasos 3 al 8 de estas pautas de diseño.

En forma resumida, el plan deberá identificar:

- Las fuentes de información que se utilizarán.
- Tipos de información que se recolectará.
- Diseño: se deberán realizar especificaciones relativas a terminales y red de comunicaciones, organización de archivos, tiempo de respuesta, hardware y software del ordenador central, fiabilidad y seguridad, etc.. Esta fase apuntará más que nada a una estructura de los costos del sistema. Y si hay más de un sistema, una determinada selección.
- Planificación: es necesario establecer unos planes detallados en términos de medios materiales y humanos, así como los recursos necesarios para que el sistema pueda funcionar correctamente.
- Estudio económico: el informe presentado a la dirección debe cuantificar al máximo los costos y beneficios que podrán obtenerse con el Sistema propuesto.

Luego de este estudio viene la decisión de continuar o no con el proyecto.

Este plan debe ser aprobado por quienes tienen la supervisión del proyecto y con posterioridad a quienes serán los futuros usuarios del mismo.

3) RECOPILACION DE INFORMACION DE LAS AREAS AFECTADAS:

El diseñador debe recolectar información de fondo sobre la organización al igual que sobre los distintos departamentos o agencias que afectará el Sistema Telemático.

También se deberá estudiar la estructura de la industria o el gobierno si es que no lo conoce.

Se deberá determinar si existen requerimientos legales o políticos de la organización que afecten el diseño futuro de la Red, pues es posible que un diseño técnicamente realizable (incluso óptimo) no sea aceptable desde el punto de vista de la organización o legal.

4) ESTUDIO DE LA INTERACCION DE LAS AREAS AFECTADAS:

En este paso, resultará conveniente confeccionar una lista de las interacciones factibles y que puedan ser predecibles, entre las áreas afectadas al proyecto y evaluar cada posibilidad hasta que se conozca la magnitud y trascendencia de dicha interacción.

Se deberá determinar la incidencia y cuantificarla correctamente para desechar las insignificantes y a atender aquellas que tengan peso en el proyecto.

5) EVALUACION Y COMPRENSION DEL SISTEMA EXISTENTE:

Resulta fundamental conocer el sistema existente, ya sea informatizado o no, lo que dará un buen punto de partida para poder realizar comparaciones e iniciar el proyecto.

Resulta interesante, en esta etapa, realizar una gráfica que muestre en forma global la interconexión de los distintos departamentos o agencias dentro de toda la estructura de la organización.

El hecho de ubicar las entidades o áreas afectadas al proyecto, determina la asignación exacta del tipo de información que será manejada (por ej. entrada / salida).

Finalmente, es necesario realizar un diagrama del flujo de información del sistema vinculando todas las estaciones o terminales.

6) DEFINIR LOS REQUERIMIENTOS DE LA RED:

La recolección y análisis de todos los elementos que permitan determinar una auténtica distribución de la información, resulta de suma importancia.

Es aquí donde deberá analizarse si es conveniente o no realizar la automatización de una determinada tarea.

Quizás pueda optimizarse dicha tarea con la sola corrección de las deformaciones más visibles y una adecuada instrucción del personal responsable de ejecutarla.

Esta situación es sumamente importante ya que si una tarea que realizada manualmente no satisface las reales necesidades dal usuario, su automatización no presentará ninguna ventaja, agregando el riesgo de hacerla más dificultosa, menos eficiente y por ende más costosa.

Debido a esto, el requerimiento de una definición exacta del problema a resolver, le permitirá al diseñador tener elementos de suma validez para encarar su resolución,.

Por lo tanto, cumplidos todos los puntos tratados hasta este momento, el diseñador deberá tener la definición del problema, toda la información de fondo necesaria, el conocimiento de las interacciones entre las áreas afectadas y una comprensión básica de los sistemas que utilizará la Red Telemática.

Aquí comienza el diseño de la Red de Comunicación de Datos.

Hay que definir, entonces, los requerimientos del sistema, y éste es uno de los pasos más críticos de todo el proyecto, ya que una mala estimación de las necesidades reales puede dar lugar a un magro resultado del proyecto y evidentemente no deseado.

Entre los recaudos más importantes a tener en cuenta están:

a) Distinguir correctamente entre requerimientos y características deseables. Por este motivo, es importante, listar todo lo que uno desea del proyecto y tenerlo bien identificado.

b) Tener como objetivo el rendimiento requerido, lo que requiere certeza, cuantificación y precisión.

c) Incluir requerimientos para:

Entradas / salidas, procesamiento, hardware / software, estructura de archivos / base de datos, portadores comunes, documentación, confiabilidad, exactitud, mantenimiento, recursos humanos, pruebas de conversión al nuevo sistema, implementación, proyección futura.

Ahora, una vez que se hayan documentado todos los requerimientos del Sistema, es fundamental realizar una presentación y análisis de los mismos a los usuarios involucrados y a la administración.

Cuando se tenga la certeza de la comprensión de lo propuesto, se puede, entonces, comenzar con el diseño físico de la Red Telemática.

7) DISEÑO DE LA RED TELEMATICA:

Como se dijo en los párrafos introductores de este trabajo, cuando estamos en presencia de una red por conmutación de paquetes, o sea la modalidad de almacenaje y envío, es necesario conocer la magnitud de los retrasos nodales y a la vez tratar de minimizarlos.

Para realizar los análisis descriptos anteriormente no resulta necesario tomar como variable los nodos de conmutación, ya que generalmente su ubicación responde a otro tipo de cuestiones, ya que se trata de Redes Públicas (población) como de Redes Privadas (actividad).

Ahora bien, definida la ubicación de nodos, deberá estudiarse la forma de conectar los mismos, lo que da lugar al estudio de la topología de la Red. Esta decisión influye en la elección de los nodos de encaminamiento que también deberán ser minuciosamente diseñados, al igual que los enlaces multipunto que pueden ser necesarios incorporar a la Red Telemática en diseño.

Se deberán involucrar en este punto el estudio y análisis de los siguientes temas:

- Análisis de los tipos de mensajes y operaciones.
- Determinación de la longitud de los mensajes.
- Determinación de los volúmenes de mensajes.
- Determinación del tráfico total.
- Determinación de la carga de tráfico en las líneas.

Retrasos nodales en las Redes Telemáticas:

Si se está interesado en representar el comportamiento de la red por un solo dato, lo más conveniente es poder estimar el tiempo medio que tarda un mensaje cualquiera en ir de extremo a extremo. La obtención del mismo se realiza a través de un tratamiento matemático que no será detallado, pero se aconseja consultar la bibliografía específica que se detalla al final de este trabajo.

Optimización de capacidades:

Cuando uno conoce el retardo medio de los mensajes entre extremo y extremo, se deberá tratar de lograr la máxima minimización del mismo.

Es evidente que cuanto mayor sea la capacidad asignada a los enlaces internodales, menores serán los retrasos pero mayor será el precio de la Red.

Existen varias alternativas para lograr la optimización de la capacidad de los enlaces, lo que no se comentará en el presente trabajo, pero se deja la advertencia de la importancia que este punto reviste al contexto general de diseño del sistema.

Diseño topológico de la Red:

Por lo general, los requerimientos de la ubicación de los nodos de una Red Telemática se conocen desde el origen del estudio del proyecto, por esto resulta muy necesario estudiar las distintas posibilidades de interconexión de los mismos, lo que determina, evidentemente, distintas topologías.

El requisito primordial a tener en cuenta resulta ser el de la "fiabilidad de la Red", es decir, su capacidad para funcionar después de haberse producido fallas en los enlaces o nodos.

Generalmente, el cálculo del flujo máximo en redes se lleva a cabo con rapidez utilizando algoritmos eficientes, que no detallaremos en este trabajo, y para los cuales el lector puede consultar excelentes bibliografías existentes.

Hasta aquí se han puntualizado y desarrollado algunos puntos importantes a tener en cuenta para el diseño de una Red Telemática; ahora desarrollaremos otros aspectos también importantes y que generalmente no se los valora de la manera correspondiente, lo cual puede generar efectos no deseados y complicaciones insalvables.

Análisis de los distintos tipos de mensajes que manejará la Red y las operaciones que deberán realizarse:

Aquí resulta necesario identificar cada mensaje mediante un título, y si es que no existe, tratar de contar con una muestra. De no existir, se lo deberá diseñar y poner a consideración de los futuros usuarios que los utilizarán.

Estos lo aprobarán o sugerirán correcciones que deberán ser tenidas en cuenta.

Luego será necesario estimar el número de caracteres que deberá asignar a cada mensaje, y con toda esta información resultará beneficioso confeccionar una tabla con dichos datos (N° de Mensaje, Nombre de Mensaje y N° estimado de caracteres).

Esta tabla debe ser conocida por los usuarios, y es aquí donde el grupo de diseño de todo el sistema deberá considerar en forma global la incidencia en la organización o empresa, dadas las preguntas que surgieran del análisis de los tipos de mensajes presentados, y puede resultar que se requiera un cambio radical de las modalidades existentes a la fecha.

Esta situación requiere un acuerdo total y absoluto entre los diseñadores, usuarios y directivos del proyecto y empresas.

Determinación de la longitud de los mensajes:

Toda la lista de mensajes incluidos en la lista realizada anteriormente debe ser correctamente evaluada, de modo de poder determinar el número promedio de caracteres para cada elemento de información en cada tipo de mensaje.

Aquí también podemos hacer una gráfica con los siguientes datos: N° de Mensaje, Contenido del Mensaje, promedio de caracteres de Mensaje y N° máximo o pico en caracteres / mensajes.

Otro análisis a tener en cuenta resulta ser el cálculo de los volúmenes de mensajes, los cuales deberán ser determinados por día y por hora, como así también el volúmen promedio diario.

Habrá que determinar si existen cargas pico de mensajes en algún horario; prever saturaciones de acuerdo a algunas circunstancias especiales como por ej. en una empresa de Aerolíneas, las variaciones estacionales, etc.

Con todos los datos anteriores el diseñador tiene elementos suficientes como para calcular el tráfico total de la red, o sea, el total de caracteres o bits por segundo en base a caracteres totales transmitidos por día y por hora.

Luego, ubicados geográficamente todos los nodos y terminales de la red, se está en condiciones de realizar un gráfico (en un mapa) indicando la cantidad de tráfico que deberá soportar cada eslabón.

Una vez calculado el total de caracteres transmitidos diariamente por eslabón será necesario convertirlo a bits por segundo, para determinar que tipo de línea de transmisión debe emplearse para conectar cada estación.

Determinación de las cargas en las líneas:

Para lograr este dato, se deberá tener en cuenta el total de caracteres transmitidos por día en cada estación de la red.
Luego se determina si existe diferencia entre horarios entre las distintas estaciones, porque esto está relacionado con la determinación de los horarios de trabajo entre extremos del sistema, lo cual permitirá optimizar la determinación de las velocidades a que se deberán transmitir los datos.

Otro parámetro importante para determinar las cargas de las líneas es el método de transmisión elegido, como así también la clave utilizada para transmitir.

Aquí también resulta necesario estudiar la proyección futura del sistema, y hacerlo en términos de entre 2 y 5 años, lo cual permitirá justificar las inversiones iniciales.

Generalmente los valores de crecimiento de los volúmenes de tráficos pueden variar entre un 10 y un 50% en este tiempo.

Deberá tenerse en cuenta también, los tiempos que introducen los errores en la transmisión de los datos (detección y corrección de errores).

Por otra parte, suele ocurrir que los usuarios, debido a su eficiencia, capacidad y rapidez, utilicen el sistema de una manera no prevista originalmente, lo cual provocará lo que suele denominarse "efecto de autopista".

Siguiendo con los factores que tienen relación con la carga de las líneas del sistema, tenemos que ver lo que es el aprendizaje de los operadores del nuevo sistema.

También influye en los tiempos de transmisión, pero que no son datos de la empresa o negocio, son aquellos como: tiempo de sondeo, caracteres de control de línea, tiempo de retorno, tiempo de sincronización de módems, tiempo de propagación del mensaje, tiempo de teclado, de impresión, de lectura, etc.

Debido a estos factores, se tendrá que aumentar la velocidad mínima a la que se transmiten los datos y prever mayores contingencias.

Con toda esta información, se deberán repasar los criterios para establecer los tiempos de respuesta del sistema.

En este momento habrá que recurrir a los textos especializados para estudiar los retrasos nodales del sistema, mediante la aplicación de la teoría del calor, encaminamiento de los mensajes, etc.

Diseño topológico de la Red:

Como se dijo antes, la localización de los nodos de la red están determinados con anterioridad por los asentamientos de la empresa o repartición, y a la vez lo mismo ocurre con los centros de conmutación cuando se trata de redes públicas, pero no obstante estas situaciones, pueden realizarse muchas maneras distintas de conexión entre nodos y terminales, lo que da lugar a diferentes topologías de la red.

Por supuesto que el hecho de buscar alternativas distintas en la configuración tiende a obtener una configuración de costo mínimo y óptimo rendimiento.

Requerimientos de software y hardware:

El hecho de utilizar distintos tipos de líneas con motivo de disminuir los costos, hace que el software tenga que ser adaptado correctamente, de modo de permitir el soporte de dicho diseño.

Además, la evaluación del software fija la cantidad de tareas no productivas que se impone el sistema, debido a protocolos, caracteres de central de línea y demás.

Y si existe una computadora central en el sistema diseñado, esto hace que deba ser evaluada la incidencia en la carga de la misma respecto a su capacidad.

Puede resultar que deban ser reconsiderados los sistemas de control de líneas, tipo de líneas, ubicación de centros multiplicadores, terminales, etc.

Se deberá ver en detalle la computadora que operará el sistema, el procesador de comunicaciones delantero que controlará las comunicaciones de datos del Sistema, los multiplicadores, concentradores, módems y terminales de cada estación.

También se deberán estudiar las redundancias necesarias de equipos con motivo de aumentar la confiabilidad del Sistema Telemático, entre los cuales se puede prever un equipo de repuesto en los puntos críticos, circuitos de repuesto, conmutación de una línea arrendada a una red pública, etc.

8) DESARROLLO DE LAS COMPARACIONES DE COSTO:

Este análisis comparativo de costos es lo que dará lugar a determinar la configuración de red más conveniente.

Los factores principales en el costo global los podemos tomar como: distancia total de cada eslabón, costo por distancia del tipo de servicio escogido y cargas por terminación de líneas (terminales de servicio).

Podemos mencionar algunos factores que deben tomarse en cuenta al realizar un análisis de costo contra beneficios para una Red Telemática.

En todas las configuraciones de red posible deberá realizarse un estudio de costo contra beneficios, y a los 2 escogidos, previo a su presentación a la administración o dirección de la empresa o repartición, se los deberá estudiar en detalle, de modo de presentar un panorama completo de ambas alternativas y, sobre todo, una correcta comprensión de la relación costo / beneficio.

COSTOS DIRECTOS:

- Equipo de computadora
- Equipo de comunicaciones
- Carga por línea portadora
- Software
- Personal de operación
- Costos por instalación (lugar, energía eléctrica, etc.)
- Reparaciones
- Mantenimiento de hardware
- Desarrollo de documentación

BENEFICIOS:

Reducciones de costos directos e indirectos

Eliminación de personal de oficina

Reducción de costos de inventario

Distribución de recursos a través de la demanda de servicio

Mejores servicios

Procesamiento más rápido de operaciones

COSTOS INDIRECTOS:

- Capacitación del personal
- Transformación de los procedimientos de operarios
- Desarrollo de software de apoyo
- Mayor tasa de salida del sistema durante el período de operaciones iniciales

BENEFICIOS INTANGIBLES:

Menor volúmen de papel, producido y manejado

Aumento en el nivel de calidad y rendimiento del servicio

Mayor capacidad de expansión

Mejoras en el proceso de decisión al proporcionar acceso más rápido a la información

Moral más alta de los empleados

9) PRESENTACION DE LA RED:

Debemos partir de la premisa de que el proyecto no debe ser solo un diseño en el papel, sino un usuario satisfecho con las bondades y performance del sistema, por esto resulta conveniente estudiar una buena estrategia para que cuando el "producto final" haya sido elaborado, sea presentado a las autoridades de la empresa o repartición, de modo tal que comprenda la filosofía, los criterios, las bondades y las problemáticas que pudieran existir.

Esto requiere que se realice una documentación extensa y clara de todo el diseño del proyecto ya que, sin duda, la misma deberá contener el siguiente detalle:

- Planteo del problema que se pretende solucionar y a la vez las consecuencias de no hacerlo así.
- Descripción de la solución propuesta.
- Análisis de costo / beneficios de las soluciones posibles, de modo que se logre apoyo a la solución propuesta.
- Plan de implementación del sistema, tiempos y costos.

Como se ve, es necesario que el informe final de presentación sea elaborado con perspicacia y conciencia del riesgo que se corre si no se maneja correctamente, ante las autoridades, los criterios y las pautas que dieran lugar al proyecto propuesto.

Resulta necesario tratar de anteponerse a las preguntas que se descontarán, seguramente, durante la exposición, tratando de dar las respuestas más adecuadas.

Para finalizar, resulta fundamental que el foco de la presentación se centre en presentar los beneficios que se introducirán con la implementación del sistema propuesto.

10) IMPLEMENTACION, SEGUIMIENTO Y REVALUACION DEL PROYECTO:

El cuidado de que todo lo proyectado se realice estrictamente, resulta una tarea primordial en la etapa de implementación del sistema. Por eso, el diseñador debe ser muy cuidadoso en este tema y evitar la incomprensión de la esencia y lógica del proyecto, lo que suele ocurrir, generalmente, con los implementadores.

Puede correrse el riesgo de cargar con la culpa de un "mal diseño" cuando en realidad las fallas están en la implementación; por este

motivo, es necesario administrar muy bien esta etapa.

Luego vendrá la etapa de pruebas y puesta a punto del sistema, lo cual es aconsejable realizarla en tres niveles, los cuales pueden ser:

nivel I : Prueba individual

nivel II : Test de módulos

nivel III: Prueba del sistema

En este momento, también se deberá evaluar el rendimiento y el funcionamiento de todas las áreas del proyecto, como por ej.: Equipo de terminal, módems, multiplicadores, concentradores, líneas de transmisión, procesadores, procesos de computadora, software, y por sobre todo, a los usuarios del sistema.

También habrá que dedicar atención a los aspectos formativos del personal que utilizará el sistema, ya que se trata de un área de vital importancia, pues el éxito final del proyecto depende en gran medida de la habilidad y capacidad para utilizar correctamente las posibilidades ofrecidas, y éstas están apoyadas en una férrea formación.

Arranque del sistema:

En la mayoría de los casos es conveniente realizar una implementación gradual y cuidadosamente planeada, teniendo en consideración aspectos humanos, técnicos y organizativos que pudieran producirse en la fase de puesta en funcionamiento.

Por este motivo, también resulta necesario realizar comparaciones entre los tráficos estimados y los tráficos reales, tratando de determinar el causante de los mismos.

Este tipo de planteos es conveniente realizarlos durante toda la vida del sistema, lo que sin duda tenderá a la optimización del funcionamiento y, por ende, a la reducción de costos.

CONCLUSIONES

Como el lector habrá podido observar, los criterios a aplicar al diseño de Redes Telemáticas son numerosos, aunque todavía queda mucho por decir.

En este trabajo hemos querido reunir, en la medida de lo posible, un conjunto de informaciones sobre el tema. El material recopilado pretende ser útil al lector interesado, y al mismo tiempo que permita sensibilizar en el tema tratado al lector ocasional, quien probablemente llegará al menos a ser usuario de un sistema telemático.

Çomo se ha visto, el diseño de redes de computadoras es uno de los problemas más complejos en el campo de la informática; como consecuencia de ello, el diseño eficiente de dichas redes es una tarea ardua en la que intervienen gran cantidad de parámetros y modos de funcionamiento. Resulta necesario entonces, introducir simplificaciones de partida, a fin de que los problemas sean analíticamente tratables.

Es aconsejable considerar las experiencias obtenidas en el proyecto y puesta en marcha de otras redes, pues se podrán detectar deficiencias y bondades surgidas en las distintas etapas de su gestación.

BIBLIOGRAFIA:

"TELEINFORMATICA Y REDES DE COMPUTADORES"

(Segunda edición)

Mundo electrónico. Varios autores bajo la coordinación de Antonio Alabau Muñoz

EDITORIAL MARCOMBO, BOIXARBU EDITORES.

Apuntes del curso "TELEINFORMATICA Y TELECOMUNICACIONES"

Dictado por el C.R.E.I. del 2 al 30 de noviembre de 1983.

Cuadernillos Técnicos de la Compañía Telefónica Nacional de España

Departamento Comercial de Telemática

Publicación de la Dirección General de Correos y Telecomunicación de España

Memoria 1982.

"CENSO IBEROAMERICANO DE RECURSOS DE INFORMACION AUTOMATIZADA"

Fundación de la red de Información Científica Automatizada

(FUINCA)

"FUNDAMENTOS DE COMUNICACION DE DATOS"

Jerry Fitzgerald - Tom S.Eason

EDITORIAL LIMUSA.

"TELEINFORMATICA"

Macchi C. y Guilbert J. F.

EDITORIAL OMEGA

"CONTROLES INTERNOS PARA SISTEMAS DE COMPUTACION"

Jerry Fitzgerald

EDITORIAL LIMUSA.

"ANALES DEL II Y III CONGRESO NACIONAL DE TELECOMUNICACIONES Y ELECTRONICA

BUENOS AIRES - ARGENTINA.

EVOLUCION DE LA FORMACION DE PROFESIONALES DE NIVEL SUPERIOR EN INFORMATICA EN CHILE

José Durán Reyes
Universidad Técnica Federico Santa María
Héctor Rodríguez Estay
Universidad Católica de Valparaíso
Chile

INTRODUCCION.

El desarrollo de la computación en Chile, se inicia en 1962 con la llegada de los primeros tres computadores. Uno de ellos, adquirido por la Universidad de Chile constituye el primer hito histórico que señala el comienzo de la influencia de la tecnología de computadores en la educación superior. Desde entonces se ha producido un crecimiento sostenido del uso de los computadores en el ambiente educacional como consecuencia lógica de la evolución tecnológica mundial y de los esfuerzos, a veces intensos, otras débiles, de las instituciones que conforman el sistema de enseñanza superior por impulsar decididamente el uso de la informática.

Para situar el ámbito de influencia es orientador relatar a grandes rasgos el grado de avance del uso de esta tecnología desde el inicio.

DECADA 1960-1969.

En la década de 1960, se inicia lentamente el equipamiento de computadores en la Universidad de Chile, Católica de Chile, de Concepción y Técnica Federico Santa María. El desarrollo de computación se gestó en Centros de Computación orientada al quehacer académico.

Inicialmente se intentó los esfuerzos en la enseñanza de lenguajes de programación, cálculo numérico y algunas asignaturas de aplicación de tecnología de computadores en áreas de ingeniería. A fines de la década se inician los primeros esfuerzos en formación de especialistas en el área (nivel de programación de computador). Lentamente también comienza el uso de los computadores en la administración de las propias actividades educacionales (matrícula de alumnos, pago de remuneraciones, selección de alumnos).

DECADA 1970-1979.

En la década de 1970 se intensifican los esfuerzos por dotar de mayores recursos computacionales al campo educacional. Sin embargo, los primeros años de esta década fueron angustiosos, dada la escasa capacidad instalada frente a las necesidades de procesamiento de datos siempre crecientes. A fines de 1974 se liberan las severas restricciones de equipamiento de computadores, produciendo un acelerado proceso de incorporación de computadores de mayor potencialidad que superó largamente el parque de

equipos instalados. Este fenómeno oportunamente captado por las Universidades de Chile, Católica de Chile, Técnica del Estado (hoy USACH) y la Empresa Nacional de Computación motivó la creación de un programa conjunto para formar aceleradamente diversos niveles de especialización, denominado Plan Nacional de Capacitación Intensivo en Procesamiento de Datos (PLANACAP). A este plan ahirieron la mayoría de las Universidades Chilenas, su interesante concepción sobrepasó las fronteras nacionales. Dejó de funcionar en 1983. En esta década se observa también ofertas de servicio de Procesamiento de Datos de las Universidades con mayores recursos computacionales, a las empresas nacionales carentes de ellos, contribuyendo en forma más directa al desarrollo informático del medio empresarial.

En el plano académico el uso de los computadores se orientó a la preparación de recursos humanos de mayor calificación, en el nivel de Ingeniería de Ejecución. Los centros de computación a su vez fortalecieron sus cuadros humanos y equipos dando respuesta a necesidades internas crecientes de procesamiento de datos de usuarios académicos y de administración.

Se debe resaltar la importancia que ya comienza a adquirir en esta década la incorporación de asignaturas de computación en los programas curriculares de otras carreras, en forma significativa las diversas ingenierías civiles, en menor grado, ingeniería comercial, ciencias básicas y algunos intentos ínfimos en medicina, leyes y pedagogías.

DECADA ACTUAL 1980-1989.

En lo que va corrido de la presente década la influencia de la computación ha continuado intensificando sus efectos en el ámbito educacional superior.

En el plano académico: la enseñanza de asignaturas de computación se ha ido extendiendo a los programas curriculares de otras carreras, y no tan sólo de aquellas del área de ingenierías. En carreras de la propia área informática la demanda de mayores recursos informáticos ha crecido notablemente como consecuencia del inicio de carreras de ingeniería civil informática que necesitan de abundantes recursos computacionales en su propio proceso de aprendizaje. En la actividad de investigación también se ha intensificado el uso de computadores en el desarrollo de proyectos de investigación, no obstante, proyectos emanados del área informática siguen siendo muy escasos.

La influencia de computación en la administración propia de las actividades educacionales se ha generalizado, no obstante persisten algunas instituciones con precario uso de computación en este sector. El procesamiento de datos se caracteriza en general por su caracter centralizado, aún; sin embargo, aplicaciones de procesamiento distruibuído y el concepto de usuario terminal comienzan a difundirse. Aplicaciones más recientes de computación como automatización de oficina, conexión a redes de computadores, acceso a bases de datos nacionales y extranjeras, técnicas de graficación, etc. recién comienzan a implementarse, con retardo ostensible, en comparación a lo que sucede en empresas nacionales de vanguardia en lo que se refiere al uso de computación.

SITUACION ACTUAL.

Es necesario destacar que entre las Universidades conviene distinguir aquellas tradicionales o creadas con anterioridad a la ley de Universidades de 1981, y las que surgieron con posterioridad a la promulgación de dicha ley. En efecto, el decreto con fuerza de ley No. 1 de 1981 del Ministerio de Educación Pública autorizó la creación de nuevas universidades privadas. Estas últimas pueden impartir las carreras que hasta entonces impartían exclusivamente las universidades tradicionales, siempre y cuando sometan previamente los programas de estudios que conducen a un título profesional a la aprobación de una entidad examinadora. También las cinco primeras promociones de las nuevas universidades deben rendir exámenes ante comisiones mixtas integradas por profesores de la entidad examinada y de la institución examinadora. Las entidades examinadoras son por lo general las universidades que existían con anterioridad a 1981. La ley de universidades dictada ese año también reconoce como de competencia estrictamente universitaria un total de 12 carreras, cuatro de las cuales deben ser impartidas como mínimo por todo nuevo establecimiento que desee el reconocimiento oficial de su calidad de universidad.

En marzo de 1985 se encontraban en funcionamiento 21 universidades, de las cuales 18 eran estatales y 3 privadas.

Las Universidades tradicionales de mayor a menor antiguedad son:

Universidad de Chile
Pontificia Universidad Católica de Chile
Universidad de Concepción
Universidad Católica de Valparaíso
Universidad Técnica Federico Santa María
Universidad de Santiago de Chile (ex Universidad Técnica del Estado)
Universidad Austral de Chile y
Universidad del Norte.

Las Universidades creadas posteriormente son:

Universidad de Valparaíso
Universidad de Tarapacá
Universidad Arturo Prat
Universidad de Antofagasta
Universidad de La Serena
Universidad de Atacama
Universidad del Bío Bío
Universidad de la Frontera
Universidad de Magallanes
Universidad de Talca
Universidad Metropolitana (privada)
Universidad Gabriela Mistral (privada)
Universidad Diego Portales (privada)

INGENIERIA CIVIL INFORMATICA.

Ingeniería (6 años de estudios) es una de las doce carreras de ámbito estrictamente universitario según la ley de Universidades de 1981 y, por lo tanto, las 21 universidades mencionadas anteriormente podrían dictarla.

La expresión "Ingeniero Civil" se utiliza en Chile con una doble acepción:

- para indicar un nivel de estudio,
- para especificar una especialidad.

La definición de un "nivel de Ingeniero Civil" fue establecido por el Consejo de Rectores de Universidades Chilenas con el fin de diferenciar a los Ingenieros con una formación de mayor contenido científico, de los ingenieros de ejecución.

Actualmente los programas de estudio para un "nivel de Ingeniero Civil" conducen primero al grado de Licenciado en Ciencias de la Ingeniería y posteriormente al título de Ingeniero Civil.

Pero recién en 1981 y 1982 hacen su aparición las carreras del más alto nivel (12 semestre de duración) como lo son las de Ingeniería Civil en Informática y Computación (3 casos) o las de Ingeniería Civil con especialidad en Computación e Informática (2 casos). Es interesante destacar que estas carreras estaban a cargo de las Universidades tradicionales, es decir, las existentes antes de la ley de Universidades de 1981.

CUADRO No. 1.

Institución	Carrera	Fecha de inicio	Duración en semestres	Vacantes
Universidad de Concepción (Concepción)	Ingeniería Civil con Especialidad en Informática	1982	12	50
Universidad Técnica Federico Santa María	Ingeniería Civil Informática	1981	12	80
Universidad de Santiago(1) (Santiago)	Ingeniería Civil en Informática y Computación	1981	12	(2)
Universidad Austral (Valdivia)	Ingeniería Civil en Informática	1982	12	60
Universidad del Norte (Antofagasta)	Ingeniería Civil Especialidad en Computación e Informática	1982	12	100

(1) En 1980, la Universidad Técnica del Estado pasó a constituirse en la Universidad de Santiago de Chile, perdiendo sus sedes de provincia.

(2) El alumno ingresa al primer año del Plan Común de Ingeniería, carrera que ofrecía 1.600 vacantes en 1982.

En 1984, a excepción de un solo caso la Universidad Católica de Valparaíso, todas las Universidades cuya fecha de creación es anterior a 1981 ofrecían la carrera de Ingeniería Civil en el área de Computación y/o Informática. A estas 7 Universidades con sus respectivas carreras se sumaba la impartida por la Universidad de Tarapacá en el extremo norte del país (1) y la impartida por la Universidad de la Frontera en Temuco (2), (ver cuadro No. 2). Ello hace un total de 9 instancias del más alto nivel (6 años de duración). Como se recordará, tan sólo cinco años antes, en 1979, aún no había sido creada ninguna carrera de este nivel. Al respecto, es interesante destacar que, así como entre 1973 y 1979 las Universidades prescinden de las carreras de nivel técnico para dar prioridad a las de Ingeniería de Ejecución, entre ese último año y 1984 las mismas entidades educacionales privilegian las carreras de Ingeniería Civil en desmedro de las de Ingeniería de Ejecución y de las técnicas, las cuales pasan a ser el campo de competencia de los Institutos Profesionales y de los Centros de Formación Técnica, respectivamente.

En lo que respecta a las vacantes, fueron cerca de 1.000 las ofrecidas en el caso de las carreras de nivel profesional, a las que hay que agregar otras 300 a 400 para incluir las ofrecidas por las carreras sobre las que no se dispone de información. Ante este hecho, es comprensible tener dudas acerca de la calidad de la preparación profesional impartida a los recursos humanos de nivel puesto que es por todos reconocida la carencia de personal suficientemente calificado para desempeñarse en funciones de docencia e investigación de nivel superior.

(1) La Universidad de Tarapacá, que comenzó a funcionar como tal en 1982, se constituyó sobre las antiguas sedes que la Universidad de Chile tenía en las ciudades de Arica e Iquique.

(2) La Universidad de la Frontera, que comenzó a funcionar como tal en 1982, se constituyó sobre las antiguas sedes de Temuco de la Universidad de Chile y la Técnica del Estado.

CUADRO No. 2.

Institución	Carrera	Fecha de inicio	Duración en semestres	Vacantes
Universidad de Chile (Santiago)	Ingeniería Civil en Computación	1983	12	(1)
Universidad Católica de Chile (Santiago)	Ingeniería Civil de Industrias Mención Computación	1984	12	(2)
Universidad de Concepción (Concepción)	Ingeniería Civil en Informática	1982	12	(3)
Universidad Técnica Federico Santa María (Valparaíso)	Ingeniería Civil en Informática	1981	12	80
Universidad de Santiago (Santiago)	Ingeniería Civil en Informática	1981	12	100
Universidad Austral (Valdivia)	Ingeniería Civil en Informática	1982	12	60
Universidad del Norte (Antofagasta)	Ingeniería Civil en Computación e Informática	1982	12	(4)
Universidad de Tarapacá (Arica)	Ingeniería Civil en Computación e Informática	1983	12	(5)
Universidad de la Frontera (Temuco)	Ingeniería Civil Industrial mención Informática	1983	12	(6)

Fuentes:

- "Guía de Ingreso a la Universidad 1984 Primera y Segunda Parte". Suplemento de El Mercurio de Santiago, noviembre y diciembre de 1983.

- "Guía de Ingreso a las Instituciones de Educación Superior Privadas. Admisión 84". Suplemento de El Mercurio de Santiago, marzo de 1984.

- Varios Anuarios Estadísticos del Consejo de Rectores de las universidades Chilenas.

(1) El alumno ingresa a Primer año común de la carrera de Ingeniería, pudiendo optar posteriormente a una de las 16 especialidades que ella ofrece. Las vacantes totales en esta carrera fueron 700 en 1984.

(2) El alumno ingresa al Plan Común de Ingeniería Civil que ofrece 330 vacantes, alcanzando a 10 las especialidades a las que se puede optar.

(3) El alumno ingresa a Primer año común de Ingeniería Civil que en total ofrece 680 vacantes y 9 especialidades.

(4) Se ingresa al Plan Común de Ingeniería Civil que dispone de 250 vacantes y 3 especialidades.

(5) Se ingresa a Primer año común de Ingeniería Civil que cuenta con 250 vacantes y 5 especialidades.

(6) Se ingresa a Ingeniería Civil Industrial que ofrece 80 vacantes y dos especialidades a las que se puede optar.

CUADRO No. 3.

CUADRO RESUMEN SOBRE EL NUMERO DE INSTANCIAS DE FORMACION IMPARTIDAS EN EL AREA DE COMPUTACION E INFORMATICA POR INSTITUCIONES DE EDUCACION SUPERIOR Y VACANTES OFRECIDAS EN PRIMER AÑO SEGUN NIVEL Y ESPECIALIDAD DE LA CARRERA 1973, 1979, 1982 Y 1984 *.

Nivel y Especialidad de la Carrera.	1973		1979		1982		1984	
	No.de carreras.	No.de vacantes.	No.de carreras.	No.de vacantes.	No.de carreras.	No.de vacantes.	No.de carreras.	No.de vacantes
1. Nivel Profesional						(1)		(2)
Ingeniería Civil en Informática (Computación)	-	-	-	-	5	(290)	7	(240)
Ingeniería Civil de Industrias con mención en Computación	-	-	-	-	-	-	2	(3)

* Se excluyen los estudios de Licenciatura en Matemáticas o en Educación con mención en Computación.

(1) No incluye las vacantes de la Universidad de Santiago cuyos alumnos ingresan a un primer año común de ingeniería.

(2) No incluye las vacantes de la Universidad de Chile, Universidad de Concepción, Universidad del Norte (Antofagasta) y Universidad de Tarapacá (Arica), cuyos alumnos ingresan a un primer año común de Ingeniería Civil.

(3) El alumno ingresa al Plan Común de Ingeniería Civil de la Universidad Católica, debiendo optar primero a la especialidad de Industrias para poder postular luego a la mención en Computación. No se dispone de información sobre la cantidad de matriculados en cada mención. Para la Universidad de la Frontera se opta a Ingeniería Civil Industrial y posteriormente a la mención Informática (80 vacantes).

Por otro lado, ante el explosivo aumento de las instancias de formación de nivel superior que ha tenido lugar en los últimos tres años en el área, cabe formularse varias interrogantes en relación a la calidad de la preparación impartida. Una de las dudas se refiere a si actualmente existe en el país un número de profesores universitarios suficientemente calificados como para impartir la docencia y si esta actividad está debidamente apoyada por actividades de investigación. Otra interrogante dice relación con la infraestructura con que cuentan los centros de enseñanza superior tanto universitarios como extra-universitarios.

Se puede constatar, que la dotación de profesores es regular en algunos casos y francamente deficiente en otros. Curiosamente, en los centros universitarios de provincias la proporción profesor/alumnos es más favorable que la registrada en la capital.

Algo similar ocurre con la proporción de profesores full-time versus los part-time, proporción que en caso de los centros de provincia varía entre un 70 y 100 por ciento mientras en Santiago desciende al 50 por ciento, aproximadamente.

Los distintos centros responsables de impartir docencia a nivel de ingeniería civil y de ingeniería de ejecución en computación e informática otorgan importancia al perfeccionamiento del cuerpo académico. Ello se refleja tanto a través de la contratación de profesionales que han obtenido algún título de post-grado en el país o en el extranjero, como del envío de una cierta proporción de su dotación al extranjero o a otras universidades del país con el fin de proseguir estudios de ese nivel o de post-título.

Considerando que la información disponible es incompleta y que pueden existir errores se incluyen los siguientes datos:

- 18 universidades e institutos profesionales que desarrollan y o aplican las ciencias de la computación en el país.

- 190 profesionales trabajan en el área de la informática, en las Universidades e Institutos profesionales, desglosados por niveles de formación.

44 Ingenieros en computación (Proc. Inf,. Sist., Inf.)
33 Master en computación (Magister Inf., M.S.C.)
6 Doctores en computación (C.S., Informática)
14 Programadores
1 Master en electrónica
1 Master en Investigación Operativa
2 Master Ingeniería Proyecto
5 Master Ingeniería Industrial
2 Master en Estadística
2 Master en Matemática
1 Doctor en Electrónica
1 Doctor en Matemática
1 Doctor en Investigación Operativa
11 Ingenieros Electricistas (Eléctricos)
17 Ingenieros Electrónicos
12 Ingenieros Químicos
5 Ingenieros Mecánicos
1 Ingeniero Geomensor
1 Ingeniero en Minas
1 Ingeniero Civil
4 Ingenieros Matemáticos
9 Ingenieros Industriales
5 Ingenieros Comerciales
1 Ingeniero Forestal

1 Constructor Civil
3 Estadísticos
6 No clasificados por falta de información.

Los principales criterios aplicados en el proceso de contratación del personal académico son la experiencia en funciones de docencia y el haber alcanzado un grado académico y, en lo posible, un título de post-grado dentro o fuera del país. En un solo caso, y solamente para los profesores full-time, se exige como requisito indispensable para la incorporación al cuerpo docente el compromiso de dedicación exclusiva. El acreditar experiencia en investigación o el tener publicados trabajos en revistas nacionales o foráneas no constituyen criterios fundamentales en la selección del personal, aunque se considera recomendable el contratar personal que haya alcanzado estos logros.

Debido a que aún no egresan de las universidades ingenieros civiles en computación e informática no se puede exigir que los candidatos ostenten un título en esta carrera. De ahí que por lo general los docentes sean ingenieros de otras especialidades (eléctricos, químicos, industriales, mecánicos), situación que en los próximos cinco años se espera habrá de variar.

La proporción del personal docente con estudios de master o doctor no deja de ser alta, fluctuando entre un 36 y un 50 por ciento del total del cuerpo académico de los centros, con predominio de los masters.

Ante la consulta acerca del equipamiento general con que cuentan las distintas unidades académicas para el ejercicio de la labor de docencia se pudo constatar que, salvo un caso de provincia, en general los centros universitarios visitados disponen equipamiento en cuanto a hardware y a software. (1) La disponibilidad de laboratorios y talleres no constituía, al parecer, motivo de grandes preocupaciones, pero el material de biblioteca, en cambio, fue considerado en la mayor parte de los casos, insuficiente.

Sin embargo, si se examina esta situación desde el punto de vista de las necesidades de las actividades de investigación, se llega a concluir que se está lejos de disponer de las condiciones materiales y técnicas mínimas para su desarrollo, lo que indudablemente atenta contra la calidad de la docencia. A este respecto, conviene señalar que, de acuerdo con estadísticas recopiladas a través de la Comisión Nacional de Investigación Científica y Tecnológica (CONICYT), se pudo comprobar que la importancia concedida a la investigación en el área informático computacional tendió a disminuir en las Universidades tradicionales en el período 1977-1983 en lugar de aumentar. La evolución experimentada por tres indicadores claves avalan esta afirmación.

(1) Esta situación probablemente se encuentra ligada al hecho de que los proveedores de equipos computacionales que operan en el país demuestran principal "preocupación" por acercar a los futuros profesionales al conocimiento práctico de sus equipos; este interés se expresa a través de la donación de equipos a los centros de enseñanza superior o de su venta en condiciones especialmente ventajosas.

En primer lugar, el número de proyectos que se encontraban en proceso de desarrollo en 1977 (es decir, podrían haber sido iniciados con anterioridad a esta fecha), era de 54, pero en los años posteriores éste apenas alcanzaba a una cifra aproximada a los 20, a excepción de 1981 que sólo llegó a 4. En el lapso de 7 años transcurridos entre 1977 y 1983, el número promedio de proyectos de investigación desarrollados en el área por siete Universidades tradicionales alcanzó así a 22.7, cifra que, sin ser despreciable, es insufienciente para generar una masa crítica de conocimientos en el área. En segundo lugar, contrastado con el número total de proyectos de investigación desarrollados por las siete universidades tradicionales consideradas en el análisis, el porcentaje de proyectos correspondientes específicamente al área computacional sólo llegó en el mismo período al 1.92 por ciento. Por último, la cifra de investigadores involucrados en proyectos en esta área evolucionó en forma similar al número de proyectos, alcanzando sólo a 67 en 1983 contra 108 en 1977.

INGENIERIA DE EJECUCION EN INFORMATICA

La formación del Ingeniero de Ejecución en el área Informática se inició en nuestro país en la década del 70. La Universidad Técnica del Estado (Santiago), inició en 1973 la carrera de Ingeniería de Ejecución en Computación e Informática de 8 semestres de duración.

Una visión cronológica de la creación de carreras de este nivel se observa en el cuadro No. 4, 5 y 6 adicionando vacantes de primer año para los años 1979, 1982 y 1983 respectivamente.

CUADRO No. 4.

Institución	Carrera	Fecha de inicio	Duración en semestres	Vacantes
Universidad de Chile (Santiago)	Ingeniería de Ejecución en Procesamiento de la Información.	1976	8	s.i.
Universidad de Concepción (Concepción)	Ingeniería de Ejecución en Computación e Informática	1977	8	60
Universidad Técnica Federico Santa María (Valparaíso)	Ingeniería de Ejecución en Sistemas de Información	1975	8	45
Universidad Técnica del Estado (Santiago)	Ingeniería de Ejecución en Computación e Informática	1973	8	200
Universidad del Norte (Antofagasta)	Ingeniero de Ejecución en Computación e Informática	1979	8	s.i.

Fuente:

- Guía Académica Universidades Chilenas 1979.

CUADRO No. 5.

Institución	Carrera	Fecha de inicio	Duración en semestres	Vacantes
Universidad de Chile (Santiago)	Ingeniero de Ejecución en Procesamiento de la Información	1976	8	s.i.
Universidad de Concepción (Concepción)	Ingeniería de Ejecución Especialidad en Computación e Informática	1977	8	50
Universidad Técnica Federico Santa María (Valparaíso)	Ingeniería de Ejecución Informática	1975	8	50
Universidad de Santiago (Santiago)	Ingeniería de Ejecución en Computación e Informática	1973	8	(1)
Universidad del Norte (Antofagasta)	Ingeniería de Ejecución Especialidad en Computación e Informática	1979	8	100
Universidad de Tarapacá (Arica)	Ingeniería de Ejecución en Computación e Informática	1982	8	60
Instituto Profesional de Santiago (2) (Santiago)	Ingeniería de Ejecución en Computación e Informática	1982	8	69

Fuentes:

- "Guía de Ingreso a la Universidad 1982. Segunda Parte". Suplemento de El Mercurio de Santiago, noviembre de 1981.

(1) El alumno ingresa al primer año del Plan Común de Ingeniería, carrera que ofrecía 1.600 vacantes en 1982.

(2) El Instituto Profesional de Santiago es un establemcimiento derivado de la Universidad de Chile. Funciona como centro de educación superior en forma autónoma desde 1981.

CUADRO No. 6

Institución	Carrera	Fecha de inicio	Duración en semestres	Vacantes
Universidad Técnica Federico Santa María (Valparaíso)	Ingeniería de Ejecución en Informática	1983	8	50
Universidad de Santiago (Santiago)	Ingeniería de Ejecución en Computación e Informática	1973	8	s.i.
Universidad del Norte (Antofagasta)	Ingeniería de Ejecución en Computación e Informática	1979	8	150
Universidad de Tarapacaá (Arica)	Ingeniería de Ejecución en Computación e Informática	1982	8	80
Instituto Profesional Santiago (Santiago)	Ingeniería de Ejecución en Computación e Informática	1982	8	77
Instituto Profesional CAMPUS (Santiago)	Ingeniería de Ejecución en Computación e Informática	1983	8	150

Fuentes:

- "Guía de Ingreso a la Universidad 1984 Primera y Segunda Parte". Suplemento de El Mercurio de Santiago, noviembre y diciembre de 1983.

Actualmente dictan esta carrera las siguientes Universidades e Institutos profesionales:

Universidad de Santiago
Ing. de Ejec. en Computación e Informática.
Universidad del Norte
Ing. de Ejec. en Computación e Informática.
Universidad de Tarapacá
Ing. de Ejec. en Computación e Informática.
Universidad Técnica Federico Santa María
Ing. de Ejecución en Computación e Informática.
Universidad Católica de Valparaíso
Ing. de Ejecución en Computación e Informática.
Instituto Profesional de Santiago
Ing. de Ejecución en Computación e Informática.
Instituto Profesional CAMPUS (Santiago)
Ing. de Ejecución en Computación e Informática.
Instituto Profesional Providencia (Santiago)
Ing. de Ejecución en Computación e Informática.
Instituto Profesional Viña del Mar (Viña del Mar)
Ing. de Ejecución en Computación e Informática.

ALGUNAS CONSIDERACIONES FINALES.

La consecuencia lógica del desarrollo tecnológico violento de los computadores, de los intentos agresivos de comercialización desde los países industrializados de la disposición abierta de los centros de consumo por conocer y emplear nuevas tecnologías, continuará traduciéndose en una penetración persistente de nuevos productos de computación, en nuestro medio, con un retardo natural en la aplicación efectiva de éstos. Algunos factores que causan este distanciamiento se refieren a la limitación de recursos económicos, a la carencia de políticas nacionales que aceleren o retarden algunas de las variables de este proceso, al déficit en el dominio de ciertas técnicas, a la ausencia de visión política de algunos directivos para impulsar más decididamente el uso de la tecnología, etc.

Las perspectivas que implica esta problemática en la educación superior se hacen más críticas a la luz de las siguientes consideraciones:

- Tendencia mundial de una mayor demanda de profesionales en el área informática, que se observa también en nuestro medio. La reacción a este fenómeno en nuestro país ha sido la creación descoordinada de numerosos programas de formación en diversos niveles, sin el soporte lógico y conveniente de adecuados recursos docentes, bibliográficos y de equipamiento computacional.

- Docentes de alta calificación transitan brevemente por el ámbito educacional succionados por el área empresarial o incluso por los propios países avanzados,complicando más aún la situación.

- La disponibilidad de mayores y mejores herramientas informáticas para apoyar los diversos niveles de la administración de la actividad educacional continuará superando las reales expectativas de su aplicación. Sin desconocer que la computación se ha incorporado como elemento importante en el quehacer administrativo, aún se aprecian algunas instituciones de educación superior con un panorama desértico en cuanto a aplicaciones de computación.

- Continuará siendo de fundamental importancia contar con políticas de alto nivel y largo alcance, que definan el rol de la informática en la organización y que orientan el desarrollo computacional general de las instituciones, abordando aspectos tales como equipamiento menor y mayor de computadores, sentido y alcance de los sistemas de información, innovación curricular considerando contenidos que aporta la informática, desarrollo de los recursos humanos que interactuan en el ambiente informático a través de oportunidades de perfeccionamiento, etc.

- Continuará emergiendo en forma expansiva el uso de micros y super microcomputadores como elementos poderosos en la innovación de las técnicas de la enseñanza. No obstante que existen aspectos positivos y negativos detrás del uso de ellos en el aula, se han generdo algunas corrientes de iniciativa privada en nuestro país para impulsar la comercialización de estos servicios. Sin embargo, la educación superior presenta un panorama casi ausente del estudio, de la investigación o del

desarrollo de aplicaciones que beneficien directamente las metodologías de enseñanza. La responsabilidad por estar al tanto de estos avances recae con mayor peso en las instituciones de educación superior encargados de preparar los recursos docentes del mañana.

REFERENCIAS BIBLIOGRAFICAS.

- Diversos Anuarios Estadísticos del Consejo de Rectores Universidades Chilenas (años 1969-1984).

- Selamé T. y Barrera M.: Recursos Humanos y Mercado de trabajo en computación e informática en Chile (Centro de Estudios Sociales, 1986).

- Durán J. y Melgarejo J.: Recursos Humanos y Computación en Chile. Creces, Santiago: mayo, vol 3 No. 5, 1982.

- Durán J. y Rodríguez H.: Informe preliminar sobre carreras profesionales en el área de informática (1986). Sub-comisión asesora de informática del Consejo de Rectores.

- Durán J. y Rodríguez H.: Estudio preliminar sobre acreditación profesional para Ingeniería Civil Informática. Comisión de informática, Colegio de Ingenieros de Chile A.G.

- Diversos Suplementos de "El Mercurio" de Santiago de Chile. Guía de Ingreso a la Universidad (años 1980-1985).

- Documentos de "Encuentros Universitarios de Computación e Informática".

REFLEXOS DA INFORMATIZAÇÃO NA SOCIEDADE

Cláudia Sabani
Universidade Federal do Rio Grande do Sul
Porto Alegre - *Brasil*

1. INTRODUÇÃO

O objetivo deste trabalho é o de fazer uma reflexão sobre o crescente uso da informática nas diversas áreas de aplicação e os efeitos que isto acarreta para o indivíduo e para a sociedade. Além disso deve-se analisar as diferenças entre o processo de informatização nos paises desenvolvidos e naqueles em processo de desenvolvimento, buscando-se as soluções adequadas para cada caso.

A informática hoje já é uma realidade: ela está presente em nosso cotidiano, e cabe a nós a tarefa de saber aproveitar seus aspectos positivos e de atenuar os danos que ela possa causar.

2. EVOLUÇÃO

A velocidade em que estão ocorrendo as transformações causadas pela informatização dificulta a adaptação das pessoas a estas mudanças.

Devemos lembrar que a primeira geração de computadores é da década de 50. Em menos de quatro décadas passamos de um computador que ocupava várias salas, com excessiva demora para realizar uma operação e altíssimo consumo de energia, cuja dificuldade de operação limitava seu uso apenas a especialistas, para os atuais microcomputadores, portáteis, adquiridos em lojas de eletrodomésticos e utilizáveis até por crianças.

Se compararmos o desenvolvimento da informatização com o de outras áreas, o contraste se torna ainda maior. Na Medicina por exemplo, os avanços são muito lentos, e o desconhecimento do próprio corpo humano ainda é muito grande. Uma outra tradicional comparação foi expressa por Peter Large em seu livro "A Micro-revolução":

"Se a indústria automobilística tivesse progredido à velocidade da indústria dos computadores, o Rolls Royce hoje custaria 5 dólares, rodaria 1 milhão de km por litro de combustível, e caberiam 4 carros na ponta de um dedo".

3. APLICAÇÕES DA INFORMÁTICA

É cada vez maior o número de áreas onde encontramos aplicações da informática. Convivemos com a informática em nosso dia a dia, numa simples operação em banco, numa compra utilizando-se o sistema de crediário, ou para reservar uma passagem aérea.

Consideraremos neste trabalho algumas áreas onde se destacam seu uso:

3.1 Aplicações da Informática na Medicina

Além dos usos administrativos como na organização do hospital ou do consultório médico, a informática desempenha importante papel:

- na análise de exames de laboratório, eletrocardiogramas, etc;
- na monitoração e observação dos pacientes;
- na administração de arquivos médicos;
- no auxílio ao diagnóstico;
- no acompanhamento de casos;
- nas estatísticas médicas;
- na construção de modelos fisiológicos e na simulação de situações para estudos;
- no auxílio em cirurgias plásticas;
- na confecção e uso de próteses;
- no auxílio à tratamento médicos e psicoterápicos.

No Brasil foi implantado pela EMBRATEL o Sistema Cirandão-Saúde, o primeiro sistema nacional de informações por computador voltado à classe médica. Os médicos de um hospital que possua convênio com o sistema têm acesso direto a bancos de dados constantemente realimentados com informações sobre pesquisas, diagnósticos e terapêuticas, além de eventos na área médica, tabelas de honorários, etc. O custo de uma consulta é o equivalente a uma ligação telefônica.

3.2 Aplicações da informática na Educação

Os usos mais frequentes do computador na Educação são:

a) Instrução Programada: o computador é colocado na posição de quem ensina o aluno. Esta é a forma mais difundida do uso de microcomputadores no ensino, tanto em escolas como em educação industrial, treinamento empresarial, etc. Porém, na maioria dos casos se limita a exercícios repetitivos, onde o computador pergunta, o aluno responde e o computador corrige. Se usado de forma criativa e bem elaborada, pode ser bastante útil no auxílio ao professor, e como um elemento de maior motivação. Mas se usado de maneira muito simples, torna o ensino maçante e desestimulante.

b) Simulação e Jogos

A simulação de um sistema permite que se teste os efeitos de várias ocorrências sobre o mesmo, e pode causar grande interesse nos estudantes, apesar de não substituir o contato direto com os fenômenos naturais.

Os jogos pedagógicos já constituem uma grande ferramenta no ensino e tem se apresentado de formas mais arrojadas com o aperfeiçoamento dos recursos gráficos.

c) Aprendizagem por descoberta

Das linguagens voltadas a auto-aprendizagem, a

mais conhecida é a linguagem LOGO, desenvolvida nos anos setenta no MIT, Massachussets, por Seymor Papert.

LOGO, além de uma linguagem, é uma filosofia da educação, que enfatiza a exploração, a investigação, devido à importância da auto-aprendizagem para o desenvolvimento das estruturas cognitivas do aluno e para uma aprendizagem mais significativa e mais agradável.

d) Pacotes Aplicativos

Apesar de não terem grande significado pedagógico, são instrumentos de auxílio úteis e muitas vezes interessantes.

É o caso dos processadores de texto, dos gerenciadores de bancos de dados, das planilhas eletrônicas.

Quanto aos processadores de textos, têm se mostrado um excelente estímulo ao ensino de uma língua, principalmente para alunos com dificuldades de redação.

Seja qual for o tipo de utilização do computador na educação é importante que ele permita a expansão da criatividade e da liberdade.

As crianças que utilizam-no como brinquedo em casa, passam a achá-lo maçante se o seu uso na escola for muito formal ou sem originalidade.

No Brasil está sendo desenvolvido o projeto EDUCOM, do qual fazem parte 5 universidades, entre elas a UFRGS. Este projeto implantou centros-piloto nas Universidades, para investigarem a utilização do computador como instrumento auxiliar no processo ensino-aprendizagem a nível de 2º grau. Tais centros reunem professores das áreas de Informática, Educação, Psicologia, Sociologia, etc, que trabalham no planejamento de atividades para a aplicação das tecnologias da Informática na Educação.

O uso da informática na Educação é bastante polêmico, em dois aspectos:

a) os benefícios/prejuízos educacionais. Já existem estudos que afirmam que o uso de computadores na Educação, se pouco cuidadoso, pode inibir as funções de memória e raciocínio das crianças. É necessário que a criança continue percorrendo todos os caminhos lógicos necessários ao seu desenvolvimento intelectual.

Existe também muita discussão em torno do que é pedagógico e do que não é; de quais as formas realmente úteis do seu uso.

b) a realidade do ensino no país e as diferenças existentes entre o ensino público e o gratuito.

No Brasil, hoje, existem 8 milhões de crianças sem escola, 27 milhões de analfabetos, 15% dos professores de 1º grau tem apenas o 1º grau, e mais de dois terços das escolas públicas não possuem qualquer recurso pedagógico auxiliar.

Aqui há um grande impasse.

Por um lado questiona-se como colocar computador em escolas que nem sempre têm quadro negro, classes e cadeiras. Por outro lado, se não dermos a chance destas crianças entrarem em contato com a tecnologia atual, estaremos aumentando sua distância das classes que já tem acesso.

3.3 Aplicações da informática na área jurídica

Esta área vem sofrendo várias transformações e muitas atividades tem sido bastante simplificadas com a introdução dos computadores.

Os extensos arquivos com normas, decretos, leis, bibliografias, etc. podem ser mais facilmente tratados se transferidos para o computador.

Grande agilização foi obtida com a automatização dos serviços dos tribunais (para controle do andamento de processos) e dos próprios escritórios de advocacia.

Já existem também aplicações onde o computador além de ser um mero acumulador de informações, auxilia no próprio processo de decisão do jurista.

3.4 Informática e Meios de Comunicação

Cada vez mais vemos exemplos do uso da informática na imprensa, televisão, publicidade, e comunicação em geral.

O desenvolvimento dos recursos gráficos vem permitindo cada vez mais sofisticações visuais e efeitos especiais, utilizados nos comerciais publicitários, na televisão e no cinema.

O uso de editores de texto, revisores ortográficos, bancos de dados vem automatizando a imprensa.

Vem sendo desenvolvido (porém pouco usado) o sistema de videotexto, onde o usuário usando um aparelho de televisão e a linha telefônica pode ter acesso a informações de bancos de dados. É portanto uma espécie de jornal eletrônico, porém com um volume muito maior de informações, e sem os problemas de impressão e distribuição dos jornais comuns.

3.5 Automação industrial

Entre todas as áreas que estão entrando na era da automação, a que mais assusta pelo perigo de desemprego e pelas modificações radicais que ocorrem, é a automação industrial.

Os principais instrumentos da automação industrial são:

a) Máquinas de Comando Numérico: equipamentos com muita flexibilidade podendo ser programados para diferentes operações.

a1) Comando Numérico Direto: um certo número de máquinas são controladas por um computador, através de ligações diretas, em tempo real.

a2) Comando Numérico Computadorizado: um microcomputador dedicado armazena e executa programas e funções básicas.

Cada máquina de Comando Numérico substitui de 3 a 5 máquinas tradicionais (com seus respectivos operadores). Porém no Brasil vem crescendo sua fabricação, o que vem a ser uma fonte de novos empregos. Em 1985, 50% destas máquinas era de fabricação nacional.

b) CAD/CAM

O CAD (Projeto Assistido por Computador) consiste de uma base de dados disponível para avaliação da integração de componentes, e usado para a execução de desenhos, plantas e projetos.

A Manufatura Assistida por Computador - CAM - é a tecnologia que compreende os equipamentos que usam o computador para controlar suas operações de manufatura.

Da integração CAD/CAM obtemos objetos projetados e manufaturados pelo controle do computador.

c) Robotização

Um robô industrial é uma máquina de comando numérico que simula movimentos humanos como os de ombro, cotovelo, pulso, dedos, cintura, etc., movimentos estes controlados sequencialmente pelo programa. Destinam-se a tarefas repetitivas, complexas ou perigosas, que requerem precisão e rapidez.

Os robôs mais simples são utilizados para operações de transferência de material; outros, mais sofisticados, operam em soldagens, pintura e montagens.

No Brasil, já são bastante utilizados na indústria automobilística.

Em termos numéricos (dados mundiais) para cada robô introduzido, três postos de trabalho são eliminados e apenas uma nova função é criada.

d) Visão artificial

O sistema funciona com a fotografia de um alvo de interesse, feita por uma câmera que a transfere digitalizada para um processador. Um programa armazenado permite então que se obtenham as informações desejadas. As principais aplicações sao guia de robôs e controle de qualidade.

e) Sistemas Flexíveis de Produção (Manufatura)

São sistemas industriais altamente automatizados, constituidos pelos elementos anteriormente descritos integrados entre si e controlados em geral por um minicomputador, que consultando a base de dados fornece às máquinas a distribuição e a seqüência de produção.

Já existem indústrias com grande produção e escasso número de empregados. E a tendência é uma ainda mais intensa substituição do homem pela máquina.

3.6 Automação bancária

A automação dos serviços prestados pelos bancos vem se expandindo muito nos últimos anos, causando grande polêmica principalmente no setor mais atingido: os bancários. Inúmeras facilidades estão à disposição dos clientes, entre elas:

- cartão magnético: permite ao usuário realizar operações de saques, depósitos, pagamentos, transferências de fundos, consultas, etc., sem preenchimento de cheques ou outros documentos.

- quiosques ou bancos 24 horas: instalados em locais de fácil acesso da população, permite aos usuários operações bancárias individuais e automáticas, com a utilização do cartão magnético (por exemplo, saques). Sua grande vantagem é o funcionamento 24 horas por dia e não apenas em horário bancário.

- terminais de consultas: para obtenção de extratos ou saldos, que são impressos e podem ser destacados pelo cliente

- terminais de compras: o cliente não precisa pagar suas compras com cheque ou dinheiro, mas sim com um débito automático em sua conta através do cartão magnético.

No Brasil o número de transações bancárias tem crescido bem mais do que o número de novos empregos gerados.

3.7 Automação comercial e automação de escritórios

São vários os instrumentos utilizados na automação de escritórios. Entre eles temos:

Sistemas de microfilmagem (integrado ao computador); processadores de texto, sistemas de mensagens eletrônicas; agendas eletrônicas; reproduções gráficas; vídeo-conferências. Estes instrumentos são empregados desde a confecção de uma carta ou ofício até os sistemas de apoio à decisão.

A automação de um escritório oferece ao profissional recursos para racionalizar em grande parte seu trabalho, liberando-o das tarefas rotineiras e tediosas.

Além disto, o treinamento para o uso de novos recursos e a eliminação de empregos, não se apresentam tão drasticamente como nos demais casos de automação.

Na automação comercial, porém, o panorama é mais intimidador. A introdução do código de barras permite uma automação de quase todas as atividades dos estabelecimentos comerciais e a previsão é de um desemprego muito grande no setor.

Além disto é muito questionável em termos de benefícios: a maior eficiência de uma loja ou super-mercado compensa todo o desemprego gerado?

4. AS CONSEQÜÊNCIAS DA INFORMATIZAÇÃO

Muito se discute a informatização contrapondo-se vantagens e desvantagens.

Para cada aspecto positivo pode-se obter um argumento para que o mesmo seja questionado; e vice-versa.

Analisaremos aqui os mais citados argumentos pró e contra a informatização.

4.1 Aspectos positivos

a) Aumento de produtividade: que deveria se refletir numa melhoria do padrão de vida do trabalhador, mas que em geral significa o lucro da empresa.

b) Melhoria das condições de trabalho com a automação:

- maior segurança pela mecanização do manuseio de materiais (o trabalhador deixa de se expor a materiais tóxicos)

- eliminação de tarefas perigosas, devido a introdução dos controles remotos, olhos elétricos, etc.

- melhoria das instalações de trabalho, que se tornaram mais limpas e organizadas.

c) Administração mais eficiente, devido a necessidade de racionalização e estruturação do trabalho.

d) Criação de novas profissões e consequentemente de novos empregos.

e) Economia de tempo; rapidez nas respostas.

f) Liberação de tarefas rotineiras e tediosas

g) Melhoria na qualidade dos produtos obtidos, devido à produção em larga escala, que impõe uniformização e precisão.

h) Independência das limitações humanas, tais como cansaço, dispersão, etc.

i) Estímulo à criatividade.

j) Facilidade de armazenamento e recuperação das informações.

k) Segurança: confiabilidade nos resultados e facilidade de preservação de arquivos de informações.

4.2 Aspectos negativos

a) Desemprego: é o item mais polêmico pela sua dimensão social. Um grande número de trabalhadores é substituído por um pequeno número de máquinas.

b) Necessidade de qualificação para novas tarefas (qualificação esta nem sempre conseguida pelo funcionário desempregado ou oferecida ao mesmo).

c) Diminuição do nível salarial da mão de obra excedente.

d) Produtos e profissões, em curto prazo se tornam obsoletos.

e) Aumento de problemas emocionais (tensão, insegurança, solidão).

f) Efeitos de maior alcance em caso de falhas técnicas ou interrupções (pois quanto mais complexo for o sistema, maiores serão os danos causados por uma falha).

g) Redução na interação dos trabalhadores, com a redução do seu número e o aumento entre as distâncias entre os locais de trabalho - a sensação de solidão é maior pelo isolamento dos companheiros.

h) Invasão da privacidade: as informações correm o risco de serem manipuladas e usadas indevidamente; além disso não se tem acesso às próprias informações e existe grande dificuldade em corrigir possíveis erros. No Brasil há uma deficiência de regulamentação sobre o assunto.

i) Crimes por computador: desde acessos indevidos a bancos de dados até movimentações ilegais de fundos de bancos, o que exige cada vez mais medidas de proteção, como senhas e criptografia.

4.3 Adaptação à mudanças

A resistência à mudanças é uma reação histórica que acontece sempre que o trabalhador enfrenta as repercussões de novas tecnologias em seu cotidiano. É pois, uma tentativa de resguardar um espaço já conquistado. A novidade passa a significar incerteza, instabilidade, principalmente quando é uma ameaça ao seu emprego e as suas relações sociais.

Não são muitas as pesquisas sobre os efeitos sociais e psicológicos causados pela informatização, pelos seguintes motivos:

- o desenvolvimento foi muito rápido e só nos últimos tempos é que se verificou a intensidade dos impactos;

- estudos deste tipo são demorados pois necessitam do acompanhamento de um grande período de tempo para apresentarem conclusões;

- a classe mais atingida é a classe operária, que não tem força suficiente para levar adiante estudos desta natureza. Aos empresários e donos de indústrias, em geral não interessam tais pesquisas.

Entretanto, alguns resultados de estudos já disponíveis nos EUA apontam problemas surgidos:

a. Os empregados deslocados de suas tarefas para outras que exigem maior responsabilidade, precisam apresentar maior índice de produção e estão cientes que qualquer falha será mais prejudicial e dispendiosa. Isto contribui para aumentar o sentimento de tensão e de pressão.

b) As alterações na organização interna das empresas, fábricas ou escritórios, transformam as relações sociais:

- a redução do número de trabalhadores diminui a comunicação entre eles, aumentando seu isolamento físico.

- mudam as relações no trabalho e a hierarquia funcional pela mudança dos critérios de promoção e avaliação dos empregados, e muitas vezes a experiência já acumulada torna-se nula;

- reduzem-se as possibilidades de acesso aos empregos, pois diminui o número de trabalhadores necessários. Isto aumenta a disputa pelos empregos e as exigências feitas pelos empregadores.

c) Com a velocidade das transformações, as profissões tornam-se obsoletas. Isto exige uma motivação maior para aprender (e desaprender) continuamente. Além disto, não são bons os efeitos causados sobre a opinião que os operários fazem de si próprios quando seus ofícios já se tornaram obsoletos.

Falta-lhes a segurança de um trabalho conhecido e de um ambiente social familiar.

d) As alterações no trabalho refletem-se também nos filhos e na família do empregado. Se a modificação for para melhor observam-se nas crianças maiores aspirações. O contrário observa-se nos filhos daqueles cujos empregos foram perdidos: são crianças mais deprimidas e cujas crenças tornam-se abaladas.

e) O maior tempo livre obtido, em geral não é bem aproveitado para o lazer, mas usado para algum outro emprego ou atividades que equivalem a um emprego.

4.4 Adequação à Realidade

A informatização vem ocorrendo tanto nos paises desenvolvidos como naqueles ainda em desenvolvimento. O que deve ser ressaltado é que as soluções válidas para os primeiros nem sempre são adequadas aos demais. Por exemplo, a automação é uma solução apropriada para paises com abundância de capital e escassez de mão-de-obra. No Brasil temos o oposto disto: escassez de capital e abundância de mão-de-obra.

É necessário pois buscar-se soluções que atendam as nossas reais necessidades.

Apesar de todas as vantagens do uso do computador numa determinada área, não se pode ignorar o contexto em que ele será utilizado.

Veja-se por exemplo o caso do uso do computador no ensino, já enfocado anteriormente. Como colocar computador em escolas onde faltam todos os demais recursos?

Também diferem os mecanismos de defesa dos trabalhadores. Nos paises desenvolvidos, a classe trabalhadora está mais protegida em relação a seu emprego e remuneração. Além disso as condições de trabalho são melhores, e existem mecanismos que garantem a estabilidade no emprego ou o re-

treinamento/readaptação do trabalhador no caso de sua função ser extinta ou modificada.

É importante que as modificações sejam planejadas com antecedência para que todos os setores afetados pela mudança tenham tempo de se restruturar. É importante que cada solução seja pensada de acordo com o ambiente onde estará inserida.

5. UMA EXPERIÊNCIA DIDÁTICA

No 1º semestre de 1986 tive a oportunidade de ministrar a disciplina "Computador e Sociedade" para o curso de Bacharelado em Ciências de Computação na Universidade Federal do Rio Grande do Sul.

Pela importância do assunto e pelas polêmicas que ele gera, penso ser fundamental introduzir-se esta discussão entre os alunos de cursos de Computação, para despertar nestes futuros profissionais a preocupação com os reflexos da informatização na sociedade.

Meus objetivos foram:

- promover uma reflexão sobre os impactos da informatização sobre o indivíduo e sobre a sociedade;

- sensibilizar os alunos para os problemas surgidos;

- questionar, na busca de soluções adequadas à nossa realidade.

Em outras universidades, esta disciplina já pertence à currículos de cursos de mestrado e doutorado, e já estão sendo feitas teses nesta área.

No decorrer da disciplina procurou-se ter em mente as diferenças do processo de informatização nos países desenvolvidos e nos países em desenvolvimento, especificamente o Brasil. Assim sendo, questionou-se as soluções adotadas pelos países desenvolvidos e sua adequação/inadequação à realidade brasileira.

As atividades do curso poderiam ser divididas em três momentos distintos:

a) Num primeiro momento, o objetivo foi essencialmente o de sensibilizar os alunos para os problemas surgidos com a informatização. Minha preocupação era a de que sendo os alunos (em sua maioria) profissionais da área de computação e que pela primeira vez tinham a oportunidade de discutir em aula este tema (apesar de já estarem em final de curso), estivessem pois voltados apenas para os benefícios da informática, tendo para si apenas uma visão do seu lado positivo. Por isto as primeiras aulas foram uma espécie de

"provocação", para despertar nos alunos ou uma atitude de defesa do que julgavam inatacável (e que seria questionado), ou um ponto de partida para discussões abertas sobre os temas propostos, entre eles: uso da informática na educação; a informatização frente à realidade nacional (pobreza); a política nacional de informática; problemas decorrentes da automação, etc... Como os assuntos tratados nesta área são sempre polêmicos, procurei oferecer aos alunos material sobre os dois lados das questões (prós e contras). As discussões eram feitas primeiramente em pequenos grupos os quais elaboravam suas conclusões e as lançavam num debate de grande grupo.

b) Com a fase anterior, de discussão em pequenos e grande grupos pretendeu-se desenvolver um espírito crítico e de questionamento. Assim, poderia-se passar para uma segunda fase da disciplina, para a qual foram planejadas palestras com especialistas da área e estudos mais aprofundados sobre os temas propostos.

c) A terceira fase do curso constitui-se do desenvolvimento de trabalhos em grupo, orientados no decorrer do semestre e apresentados ao final do mesmo. Os temas escolhidos foram: aplicações da informática nas diversas áreas (educação, ciências, administração, etc.); impactos de informatização sobre o emprego; Política Nacional de Informática e Indústria Nacional de Informática; Automação Bancária; Automação Comercial; Automação Industrial e Automação de Escritórios.

Durante todo o curso pretendeu-se examinar com clareza e sem mistificações toda a problemática dos efeitos da informatização da sociedade. Procurou-se analisar todas as vantagens trazidas tais como: maior produtividade, melhoria nas condições de trabalho, economia de tempo e liberação de tarefas tediosas, melhoria na qualidade dos produtos obtidos, estímulo à criatividade e a racionalização das tarefas, etc. Porém não foram esquecidos os aspectos negativos: eliminação de empregos, pela substituição homem/máquina; invasão da privacidade: diminuição do nível salarial da mão-de-obra excedente; produtos e profissões obsoletos a curto prazo, etc.

Concluindo, considero que os objetivos da disciplina foram atingidos pois houve uma conscientização sobre a importância de tais questões, houve um questionamento aberto e franco sobre os problemas nacionais na área da informática e uma busca de soluções que atendam às nossas reais necessidades.

6. CONCLUSÃO

Mesmo sendo a informatização um processo irreversível, ela continua como fonte de grandes discussões.

Por um lado temos a imposição competitiva, que obriga a adoção de novas tecnologias para a obtenção de maior produtividade; por outro lado temos os problemas sociais decorrentes da substituição do homem pela máquina ou da modificação das profissões e da estrutura do emprego.

Se o desemprego decorre da introdução de novos equipamentos automatizados que tomam o lugar dos trabalhadores, ele também poderá ser causado pela não adoção das inovações tecnológicas que acarretarão a perda da competitividade e do mercado, e consequentemente a demissão de trabalhadores.

De um modo geral não é a validade da informatização que vem sendo contextada, mas sim a forma como ela vem sendo implantada.

Os trabalhadores brasileiros não são contra o progresso tecnológico desde que possam participar desta evolução e ser preparados para enfrentar as mudanças.

Em outros paises são usados mecanismos de proteção ao trabalhador tais como:

- mudanças futuras nas funções já devem constar no contrato de trabalho;

- implantação gradativa (em prazos combinados) da automação para evitar o desemprego;

- garantia de salário por período de tempo determinado, dentro do qual o trabalhador é treinado para uma nova função.

No Brasil os trabalhadores vem lutando para obtenção de algumas melhorias, tais como:

- redução de jornada de trabalho;
- distribuição dos ganhos;
- melhores condições de trabalho e de vida;
- retreinamento, qualificação e deslocamento para novas funções ao invés do desemprego.

Isto pode ser viabilizado através de negociações entre empregados e empregadores e também através de legislação adequada.

A evolução tecnológica deve ser pois acompanhada de avanços na área social. O objetivo final de toda e qualquer inovação deve ser a melhoria da vida do indivíduo e da sociedade.

BIBLIOGRAFIA

LARGE, Peter. A microrevolução

MARTIN, James. Computador, Sociedade e Desenvolvimento.

TAVARES, Cristina. Informática - a batalha do século XXI

EINZIG, Paul. As consequências econômicas da automação.

ARNSTEIN, George e outros. Panorama da automação

BARELLI, Walter. Informática, educação e trabalho.

Anais CONAI 85.

Revista Dados e Idéias. Out/85.

Revista INFO. Nov/85.

Revista VEJA. 12/02/86.

ESPECIFICACIONES ALGEBRAICAS DE TIPOS ABSTRACTOS DE DATOS PARA UN CURSO MEDIO DE PROGRAMACION

Alvaro Tasistro
Alfredo Viola
Facultad de Ingeniería
Montevideo - *Uruguay*

1.-Antecedentes.-

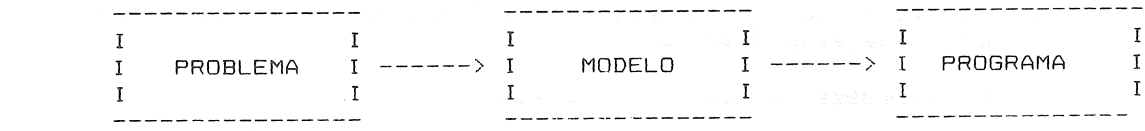
A partir del segundo semestre de 1985 comienza a aplicarse un Plan de Ajustes de Contenidos a las materias de la carrera de Ingenieria en Sistemas de Computacion, ideado para contrarrestar parcialmente el estancamiento de mas de diez anos en el desarrollo universitario de esta area de conocimiento provocado por la politica aplicada durante el regimen de facto.

Este Plan de Ajustes guio la elaboracion de nuevos cursos de Programacion basandose en un enfoque ingenieril de la disciplina que se describe a continuacion.-

1.1.- Orientacion General de los cursos de Programacion

Se visualiza la programacion como un proceso de transformacion de enunciados informales de problemas en programas aptos para ser ejecutados en computadores.

Una simplificacion esquematica podria ser la siguiente (AHU83):



En el proceso de transformacion de problemas enunciados originalmente de manera natural-informal a programas para computador se considera una etapa de modelacion o especificacion formal, con las siguientes caracteristicas:

- Se genera un enunciado del problema en terminos de propiedades consideradas relevantes de los objetos presentes en la proposicion original y no en terminos de los objetos relativos al computador o lenguaje de programacion en que se resolvera el problema. En este sentido, la especificacion formal es de "alto **nivel de abstraccion**".

- Es un enunciado del problema en un lenguaje formal, caracteristica que le es comun con el lenguaje de programacion final y que lo distingue del lenguaje natural ambiguo. Esto, ademas de proporcionar rigor al enunciado posibilita la verificacion formal de la adecuacion de la implementacion final.

Idealmente podria visualizarse la programacion como la modelizacion (simplificacion - formalizacion) de una situacion propuesta (problema), seguida de su transformacion en un enunciado equivalente logicamente, escrito en un cierto lenguaje de programacion.

Segun esta vision (extrema) el esfuerzo central estaria

concentrado en la especificacion del problema en tanto la codificacion de programas tenderia a convertirse en un proceso mecanico.

Aunque existen lineas de investigacion que se desarrollan en ese sentido (iniciadas por M&W75, B&D77), este extremo no se ha alcanzado.

La tendencia, sin embargo, es evidente.

Hay una evolucion desde la etapa en que el esfuerzo central de la programacion residia en la codificacion de algoritmos y estructuras de datos "ad-hoc", manejando elementos de bajo nivel (la obra de Knuth (Knu73) puede ser considerada la culminacion de aquel estado del arte) hacia una nueva forma de trabajo donde los elementos manejados son de un nivel de abstraccion comparable al de los enunciados naturales y donde la generacion de algoritmos y estructuras de datos puede, si no mecanizarse, al menos convertirse en la adaptacion de soluciones ya bien estudiadas y clasificadas (ejemplo claro de lo cual es el texto de Aho et al (AHU83)).

Esta tendencia tiene tambien efecto sobre los lenguajes de programacion. Estos recorren un camino que va en direccion de los problemas, incorporando elementos de nivel de abstraccion creciente e intentando desprenderse del caracter procedural de los lenguajes originales.

La constatacion de esta evolucion y la necesidad de adaptarse a ella inspiraron la elaboracion de los nuevos cursos de Programacion de esta carrera. En particular, en el paragrafo que sigue se comentara la situacion del curso "Programacion III" en el cual se incluyeron los temas relativos a Tipos Abstractos de Datos que son motivo de este trabajo.

1.2.- Antecedentes del curso "Programacion III".-

Los estudiantes de este habian tomado previamente un curso de Programacion inicial donde se desarrollaba la perspectiva senalada en el paragrafo anterior, basicamente en dos lineas:

- Estudio de los elementos de los lenguajes de programacion algoritmicos de alto nivel (Pascal fue el mas referenciado, aunque ningun curso de la carrera se basa en la ensenanza de un lenguaje especifico).

- Elementos de estilo y tecnicas de diseno de algoritmos.

Dentro de esta segunda parte se hacia enfasis en el diseno de modelos de datos e inclusive se manejaban ideas de tipos abstractos de datos. Sin embargo, por tratarse de un curso de iniciacion, ningun lenguaje de especificacion era definido.

Las especificaciones se hacian en una variedad de ellos, siendo los mas utilizados algunas derivaciones de la logica de predicados y extensiones "ad-hoc" de Pascal.

El curso "Programacion III", considerado de nivel intermedio, debia ocuparse del estudio de estructuras de datos y algoritmos, con especial atencion a los aspectos de complejidad.

De acuerdo a la orientacion general ya senalada, se opto por seguir el texto de Aho et al (AHU83) que le es razonablemente fiel, complementado por la presentacion de un lenguaje formal de especificacion para tipos abstractos de datos, a saber: Especificaciones Algebraicas de Tipos Abstractos de Datos (en adelante, EATAD).

2.- Presentacion de EATAD.-

2.1.- Forma de elaboracion.-

El diseno y preparacion de los cursos del primer semestre lectivo de 1986 se realizaron conjuntamente por los docentes encargados y grupos de estudiantes que colaboraron honorariamente. Esto permitio la realizacion de seminarios donde los temas del curso eran parcialmente expuestos, con el objetivo de identificar los conceptos de mas dificil comprension.

La mayor dificultad en relacion a las EATAD residio en la ausencia de textos donde el tema se dearrollara didacticamente. Los materiales disponibles eran articulos donde las investigaciones originales eran publicadas.

Esto condujo a la decision de elaborar el material que se incluye en la seccion 4 de este trabajo. Las discusiones en los seminarios permitieron definir la estructura de esta presentacion, la cual comentamos brevemente en el siguiente paragrafo.

2.2.- Generalidades sobre la presentacion.-

La presentacion se estructura en tres partes:

i) Una introduccion a los objetivos que la tecnica persigue, esencialmente: la independencia de la implementacion, que permite mayor generalidad en la especificacion y mayor amplitud en la eleccion posterior de implementaciones. Aqui, se trata de justificar como las caracteristicas de las EATAD (basicamente, definicion de objetos por su comportamiento frente a determinadas operaciones) conduce a la independencia de representacion pretendida.

ii) Presentacion formal del lenguaje de EATAD.
Se da aqui el marco matematico (algebras heterogeneas y de tipo) (G&H78) y luego se concentra el desarrollo en la definicion por clausura del espacio del tipo, punto que merece especial atencion, y en el estudio de las operaciones selectoras, como portadoras de la semantica del tipo definido.

iii) Una guia metodologica para el diseno de EATAD.
Es una simplificacion del algoritmo de (G&H78) que garantiza la completitud-suficiente de la especificacion. No se incluye el desarrollo exhaustivo del mismo ni las pruebas relacionadas por exceder los prerrequisitos teoricos del curso.

3.- Conclusiones.-

Las conclusiones sobre el aprovechamiento del tema por parte de los estudiantes no pueden ser completas debido a que, a la fecha, la unica herramienta de evaluacion masiva disponible (examen final) no ha sido llevada a la practica.

Sin embargo, algunas impresiones parciales recogidas durante el curso pueden ser resenadas:

- Los estudiantes parecieron atraidos por el tema. Las ideas de diseno desarrolladas en 1.- son, en general, comprendidas y puestas en practica.

- Se detecto la necesidad de un mayor desarrollo de la teoria relativa a EATAD para cubrir problemas complejos donde aparezcan funciones con efectos secundarios. Estos parecen ser de dificil resolucion, sobre todo para personal habituado a lenguajes algoritmicos. Esta clase de funciones debe ser especificada ya sea por la definicion de TAD estructurados o por una descomposicion adecuada en funciones elementales. Cualquiera de las alternativas no es, generalmente, trivial.

- El resto del curso cubrio el estudio de implementaciones (estructuras de datos y algoritmos). Alli se detectaron problemas para derivar implementaciones a partir de las EATAD. En especial se noto la tendencia a considerar la EATAD como un algoritmo pasible de ser directamente implementado, lo cual a menudo conduce a soluciones totalmente ineficientes.

Bibliografia.-

AHU83 : Aho A., Hopcroft J., Ullman J.
Data Structures and Algorithms
(Addison Wesley, 1983).

B&D77 : Burstall R., Darlington J.
A Transformation System For Developing Recursive Programs
(JACM 24(1), 1977).

G&H78 : Guttag J., Horning J.
The Algebraic Specification Of Abstract Data Types
(Acta Informatica 10, 1978).

Knu73 : Knuth D.
The Art Of Computer Programming (vol. 1, 3)
(Addison Wesley, 1973).

M&W75 : Manna Z., Waldinger R.
Knowledge and Reasoning in Program Synthesis
(AIJ 6,2, 1975).

ANEXO

ESPECIFICACION ALGEBRAICA DE TIPOS ABSTRACTOS DE DATOS

Autores: Alvaro TASISTRO, Alfredo VIOLA.

0- RESUMEN Y OBJETIVOS

Este trabajo tiene como objetivo servir como texto de apoyo al curso teorico de Programacion III para el desarrollo del tema "Especificacion algebraica de Tipos Abstractos de Datos". Para la elaboracion del mismo se consultaron diversos trabajos originales, asi como aportes metodologicos extraidos de los seminarios previos al curso.

Reconocemos la colaboracion estudiantil en la edicion del trabajo, asi como en las discusiones metodologicas para la mejor presentacion del tema.

Se presenta la tecnica de especificacion algebraica de tipos abstractos de datos (en adelante TAD) a traves de un analisis en que progresivamente se profundizan los conceptos propios de esta tecnica (secciones 1 y 2). En particular se da en 1 una caracterizacion informal de los objetivos perseguidos con la especificacion algebraica de TAD y en 2 su presentacion formal. Se culmina en 3 con pautas generales que guian el disenio de este tipo de especificacion.

La presentacion hace uso de conceptos estudiados en matematicas, la mayoria de los cuales pretenden ser explicados en el propio desarrollo. Sin embargo, es obviamente necesaria la familiaridad del lector con conceptos elementales de teoria de conjuntos, y la definicion axiomatica de los naturales y recomendable el poseer nociones de la teoria de estructuras algebraicas.

Las definiciones y citas fueron extraidas del trabajo original de (G&H78),salvo en los casos en que la fuente se indique expresamente.

1- QUE ES UNA ESPECIFICACION ALGEBRAICA DE TAD? (I: CONCEPTO)

1.0- RESUMEN:

Aqui damos una aproximacion intuitiva al concepto de especificacion algebraica de TAD por la via de presentar los objetivos que esta persigue y el modo en que propone alcanzarlos (razonablemente justificados).

Sobre el final, presentamos un primer caso concreto de especificacion algebraica: el STACK (de venerable tradicion como ejemplo de TAD) y comentamos las ventajas que pueden obtenerse al aplicar este tipo de tecnica.

1.1- DEFINICIONES DE OBJETOS Y NIVELES DE ABSTRACCION.

Podemos manejar un sinonimo mas conocido para especificacion: definicion.

Esperamos que la mera mencion de esta palabra sea, a los efectos de comprender el concepto, suficiente. No nos introduciremos en la riesgosa tarea de definir "definicion", pero si profundizaremos en la idea a traves de un ejemplo:

Supongamos que sea necesario definir (especificar) un objeto sumamente conocido y comun, por ejemplo: un vehiculo automotor. Lo mas normal en estas circunstancias (o al menos lo que ha sucedido a quienes suscriben) es formarse una imagen visual particular (o eventualmente, mas de una, en sucesion) de este objeto que se desea caracterizar. De hecho, la presencia de esta imagen es el antecedente inmediato de la definicion, lo cual puede condicionar su generalidad.

Nuestra intencion es obtener las definiciones (especificaciones) lo mas abstractas (menos dependientes de una relizacion particular) y por lo tanto lo mas generales que sea posible. Si bien podemos definir el objeto vehiculo automotor por la via de recorrer imagenes (en general visuales) de realizaciones particulares, preferiremos otra aproximacion: evitar el riesgo de comprometernos con casos particulares (coches, motos, camiones, omnibus, etc) y extraer de todos estos los COMPORTAMIENTOS COMUNES que nos importen. En un sentido, elegiremos una abstraccion funcional, en cuanto NO MANEJAREMOS IMAGENES DE VEHICULOS SINO SUS COMPORTAMIENTOS FRENTE A DETERMINADAS OPERACIONES. Habra operaciones que permitan verificar propiedades del objeto (en nuestro caso: ?esta encendido?, ?esta en movimiento?, ?que trayectoria sigue?) y otras que alteren estas propiedades (encender, arrancar, frenar, apagar).

En nuestro ejemplo, trataríamos, por esta via, de definir vehiculo automotor sin hacer referencia a formas o imagenes, sino apenas a comportamientos.

Esto independiza de las realizaciones particulares estableciendo apenas aquellas operaciones que pueden ser aplicadas al objeto en cuestion.

1.2- EL STACK

Vayamos a un ejemplo mas familiar en el mundo de la computacion: el STACK. Trataremos de evitar las imagenes de representaciones particulares (array, listas con punteros, aun mas: la idea de secuencia) prefiriendo, por el contrario, la via de listar las operaciones que realizaremos sobre el tipo y los efectos que estas provocan.

En un sentido ,algo similar ocurriria si estuvieramos hablando en terminos de un lenguaje de programacion en que STACK estuviera predefinido: no nos dariamos cuenta de su forma,sino de las operaciones que podriamos aplicarle y de sus posibles resultados.

Una especificacion algebraica para el tipo STACK (de naturales) seria como la siguiente:

```
        | New:---->STACK
        | Push:---->STACK X N ---->STACK
    1 (
        | Pop:STACK---->STACK
        | Top:STACK---->N


        | Pop(New)=indefinido
        | Pop(Push(s,i))=s
    2 (
        | Top(New)=indefinido
        | Top(Push(s,i))=i
```

No queremos profundizar aqui en el estudio de esta especificacion algebraica. Basta observar que se definen en 1 los dominios y recorridos de 4 funciones: New, Push, Pop y Top.

Son las operaciones aplicables al tipo Stack. Los efectos de estas se vinculan entre si en 2.A la parte 1 la llamaremos especificacion sintactica y a la 2, los axiomas (o especificacion semantica) del tipo.

Esta declaracion establece el comportamiento esperado del tipo Stack. En un lenguaje dado (ejemplo:Pascal) mas de una realizacion concreta seria posible y puede elegirse en un proceso posterior.Al separar especificacion de representacion (implementacion) logramos:

A) dividir un gran problema en otros menores. Ejemplo: programar funciones de Stack se separa en definirlas formalmente y elegir implementaciones.

B) al ser formales la especificacion y la version final de la implementacion es posible

verificar que la segunda sea rigurosamente correcta.

C) El problema de elegir una implementacion puede atacarse en profundidad con el objetivo de administrar lo mas eficientemente posible los recursos disponibles .

El punto central esta en que por esta via se logre trasladar el esfuerzo de diseno a la etapa de generacion de la especificacion formal, desplazandola desde la etapa de codificacion.

El trabajo de programacion era, originalmente, un esfuerzo que requeria, a la vez, la comprension del problema estudiado e inventiva para la creacion de algoritmos y su codificacion para una maquina particular.

La idea es separar el enunciado preciso del problema (su especificacion) de su resolucion para un ambiente de programacion particular (implementacion). En la primera etapa podrian, idealmente, ignorarse las caracteristicas del computador o lenguaje de programacion final. En la segunda, no deberian existir problemas de comprension del problema originalmente enunciado.

El costo total del diseno disminuiria desde el punto en que el lenguaje de especificacion es de alto nivel de abstraccion lo cual abarata la tarea de modelacion y que, una vez ella este concluida, su traduccion al lenguaje de programacion final no requiere de esfuerzos adicionales de comprension y de inventiva.

La ventaja de esta aproximacion es especialmente notable cuando nos enfrentamos a problemas de gran porte. En estos casos, mezclar discusiones sobre implementacion (como hacer las cosas?) con discusiones sobre especificacion (que hay que hacer?), suele conducir a productos que, con enorme eficiencia en terminos de tiempo de maquina, resuelven problemas que no tienen nada que ver con el propuesto y son dificiles de corregir.

2.- QUE ES UNA ESPECIFICACION ALGEBRAICA DE TAD? (II: Forma)

2.0.- RESUMEN:

Aqui presentamos una base formal para desarrollar los antecedentes intuitivos de 1. En particular se aplica la rama del Algebra que estudia las estructuras algebraicas. En la literatura estas son llamadas "algebras", lo cual no implica que se este hablando de la estructura particular llamada algebra.

Con este marco, se estudian propiedades de las operaciones definidas para los tipos en cuestion y se dan orientaciones generales para el diseno de especificaciones.

2.1- TIPOS COMO ALGEBRAS:

El ya conocido concepto de TIPO DE DATOS (un dominio de valores y un conjunto de operaciones aplicadas sobre el dominio) es enunciable formalmente con herramientas matematicas analogas: las ESTRUCTURAS ALGEBRAICAS o ALGEBRAS.

Conviene prestar atencion a la terminologia,que usualmente esta cargada de ambiguedades:

Un ALGEBRA HOMOGENEA es una pareja (C,F), donde C (dominio) es un conjunto no vacio (de valores) y F un conjunto finito de operaciones tales que cada operacion Fj es una funcion $Fj: C^n \longrightarrow C$

Observar que todas las operaciones se aplican a n-uplas tomadas de un unico conjunto (el C). Esta definicion no incluye estructuras como la del STACK, en la cual existen operaciones con dominios y recorridos en mas de un conjunto. De aqui surge la definicion (mas adecuada a nuestras intenciones):

Un ALGEBRA HETEROGENEA es una pareja (V,F), donde V es un conjunto de conjuntos no vacios Vi y F un conjunto de funciones Fj, $1 \leq j \leq m$ tales que $Fj: Vi1 \times Vi2 \times \ldots\ldots\ldots \times Vin \longrightarrow Vk$, donde Vih pertenece a V, h= 1,2,.....,n y Vk pertenece a V

Analicemos la nueva definicion. Recordando que queremos definir operaciones que involucren mas de un conjunto (ejemplo: stacks y naturales) la forma de estas operaciones debe cambiar, dejando de ser:

$$Fj: C \xrightarrow{n} C$$

para permitir componer en el dominio a mas de un conjunto.

Asi, al contrario de un algebra homogenea en que las operaciones involucran a un unico conjunto C, en las heterogeneas se definen sobre un conjunto de conjuntos (V). Cualquiera de estos puede tomarse tambien como recorrido de las operaciones definidas .

Con esto, el planteo gana generalidad y es posible representar funciones como PUSH : STACK x NATURAL ----> STACK. Sin embargo, rapidamente observamos que la generalidad que hemos ganado puede resultar excesiva.

Recuerdese que estamos desarrollando una herramienta para describir formalmente el comportamiento de una determinada clase de objetos (los objetos del tipo abstracto en cuestion). El poder de las algebras heterogeneas permite incluir operaciones que describen comportamientos de mas de una clase de objetos, relacionandolos mutuamente.

En nuestro ejemplo, el conjunto V tiene como elementos al conjunto de los Stacks y al de los Naturales. Aplicando la definicion de algebra heterogenea, podria existir en F una operacion como:

Sum : N X N --> N (suma de naturales).

Es obvio que esto es redundante si nuestra intencion es definir el TAD STACK. La suma de naturales no ayuda a describir propiedades de los STACKs, sino de los propios naturales.

Las algebras heterogeneas son un formalismo apropiado para definir a la vez a los STACKs y a los naturales. Si, por el contrario, se pretende hacer corresponder una de estas estructuras a la definicion de un unico tipo, se tienen dos consecuencias:

- Aquellos tipos distintos del que se pretende definir y que intervengan en esa especificacion deben ser considerados predefinidos, o definidos por separado.

  Ejemplo: Se pretende definir STACK de naturales.
  El tipo Naturales se considera definido a los efectos de la especificacion de STACK.

- Conviene restringir la definicion de algebra heterogenea para adaptarla a esta situacion, definiendo un nuevo tipo de estructura: el algebra de tipo.

| Un ALGEBRA DE TIPO es un par (V,F) donde V es un conjunto de con- |
|juntos Vi, 1<=i<=m, no vacios, uno de los cuales es distinguido y se|
|denomina TOI (tipo de interes o type of interest en ingles), y F es |
|un conjunto finito de operaciones Fj tales que: |
| Fj:Vi1xVi2x....xVin----->Vk, |
|donde Vih pertenece a V, con1<=h<=n, Vk pertenece a V |
|y al menos uno de los conjuntos Vi1,Vi2,..Vin,Vk es el TOI |

Hay que apreciar el nuevo concepto introducido: estamos modificando apenas una condicion de las operaciones del algebra exigiendo que el TOI (el conjunto de los objetos cuyo comportamiento queremos definir) participe (como parte del dominio o como recorrido) en todas las operaciones.
La ventaja de esta definicion frente a la definicion concurrente de mas de un tipo, implicada por las algebras heterogeneas, es la de la nitidez ganada. Efectivamente, construir objetos mas complejos a partir de otro cuya estructura es, o bien conocida, o bien se define separadamente, es una practica tipica en la programacion y que coherentemente, aplicamos aqui. Frente a ella, la tecnica de mezclar propiedades y comportamiento de indole diversa resulta superada por engorrosa.

2.2- SOBRE LOS DOMINIOS DE LOS TIPOS.

2.2.1- COMO DEFINIRLOS?

Hemos dado un formalismo para especificar TAD: los algebras de tipo.
Nuestro objetivo debe ser ahora dar metodologias o al menos orientaciones para construir este tipo de especificaciones. Con esta perspectiva estudiaremos ejemplos con el fin de inducir propiedades generales interesantes. Retomemos el caso STACK de NATURALES (dado en 1):
Deberiamos poder reconocer en esta especificacion, los elementos del algebra de tipo correspondiente. En particular concentremonos en el conjunto V, cuyos elementos son los dominios de los tipos involucrados en las operaciones definidas.
Podemos establecer inmediatamente que V = (STACK, N), es decir el conjunto de dominios incluye el conjunto de los STACKS de NATURALES y el de los NATURALES, donde el TOI es obviamente STACK.
No obstante, haber adjudicado un nombre a cada conjunto interviniente no alcanza para definirlos. Si bien podemos parecer satisfechos con la simple mencion del nombre NATURALES o de su simbolo mas comun "N" , esto no constituye una definicion.

En este caso particular esos nombres evocan un conjunto ya conocido y sobre cuya definicion no vale la penainsistir.

No nos preocupamos entonces demasiado por los naturales, sabiendo que existe una forma de definirlos rigurosamente, que, sin embargo, no es sustancial en este momento. El caso no es igual para los *STACKS*.

El nombre *STACK* no evoca ninguna definicion anterior, ya que en este momento se supone estamos especificando el TAD *STACK* por primera vez.

No hay siquiera un conjunto que puede ser tomado como base, ni simbologia apropiada para los *STACKS*. Por tanto tenemos que dar una DEFINICION RIGUROSA de este conjunto.

Ahora bien, en este punto en que comprendemos la necesidad de definir expresamente el dominio del tipo, deseamos recordar nuestras intenciones en relacion a la forma de la especificacion. Por lo expuesto en 1, queremos conseguir que nuestra especificacion sea independiente de la representacion de los objetos definidos. En particular tratamos de definir estos objetos no a traves de caracterizar su forma sino su comportamiento frente a determinadas operaciones. Esta idea, aplicada al caso que estamos estudiando, conduce a una tecnica de definir conjuntos que, en abstracto, podria caracterizarse asi:

"El conjunto X esta formado por todos aquellos valores que sean resultados del conjunto de operaciones Cx".

La idea sera definir ciertas operaciones y establecer que los resultados obtenidos por todas las aplicaciones posibles de estas operaciones son los elementos del conjunto a definir .

Esta idea esta en la base de la definicion axiomatica de los naturales. En ella se dice :

1) El cero es natural
2) Hay una funcion, (succ) que aplicada a un natural devuelve un natural.
3) Los naturales son solo los objetos descritos en 1) y 2).

Hay que puntualizar que esto es apenas una parte de la definicion de los naturales, la cual completaremos luego. Sin embargo, ya obtenemos de aqui conclusiones valiosas .

Podemos definir un conjunto C a partir de un conjunto G y un conjunto de operaciones F como sigue:

1) x pertenece a G -----> x pertenece a C
2) Fj pertenece a F y x1, x2, .., xn pertenece a C ---->
   -----> Fj (x1,....xn) pertenece a C
3) Esos son los unicos miembros de C

C se llama la "clausura de F sobre G" (Cl(G,F)), G se llama "el generador de C". En el caso de los naturales, podemos tomar como generador al conjunto que contiene al cero y F contiene una sola operacion: SUCC.

Entonces, siguiendo el esquema anterior, definimos los naturales: N = Cl((0), (SUCC) de donde :

por 1) 0 pertenece a G -----> 0 pertenece a N
por 2) 0 pertenece a N ----->succ(0) pertenece a N , etc

Con este esquema de definicion por clausura nos acercamos a la idea de caracterizar todo el dominio a traves de operaciones. En el caso de naturales, decimos que obtenemos los elementos del dominio por sucesiva aplicaciones de la operacion SUCC, salvo uno de ellos, que suponemos existente y que es el cero.

Sin embargo, un recurso mas puede ser usado para que tambien el cero sea resultado de una operacion, la cual debera comportarse como constante y, por lo tanto, no necesitara argumentos. De este modo definimos la operacion CERO.

Hecho esto, no necesitamos que G contenga ningun valor. De acuerdo al esquema de definicion por clausura, los naturales se obtendran componiendo de todas las formas posibles a las dos funciones: CERO Y SUCC.

En otras palabras: N= Cl (( ), (CERO, SUCC)) y el conjunto generador pasa a ser vacio pues el antiguo elemento primitivo (cero) puede ahora ser obtenido aplicando una operacion (CERO), no necesitandose suponer su existencia.

Como se traduce esto en una especificacion algebraica?. En el caso de los naturales escribiriamos:

CERO: -----> N
SUCC: N ----> N

Estas dos clausulas contienen todas las ideas manejadas hasta aqui. Estan estableciendo:

Los elementos de N se obtienen por la clausura de CERO y SUCC sobre el conjunto vacio. En otras palabras: CERO produce un natural (vease que es funcion sin argumentos, o sea, constante).

SUCC(CERO) es un natural, pues es la aplicacion de SUCC a un natural. Ahora: SUCC(SUCC(CERO)) lo sera tambien, etc.

Es claro que esto no completa la definicion de los naturales. Otros axiomas estan todavia omitidos. Mas adelante se vera como enunciarlos.

En el caso de STACKS de NATURALES, existen tres operaciones que cumplen el papel de CERO y SUCC:

New: ----> STACK
Push: STACK x N ----> STACK
Pop: STACK -----> STACK

Notar que en esta clausura intervienen dos conjuntos (STACKS y N) de lo cual proviene la necesidad de la siguiente definicion :

Dada una familia de generadores Gi y una familia F de funciones, se define una familia de conjuntos Ci como:

1) x pertenece a Gi --) x pertenece a Ci

2) Fj : Vi1 X Vi2 X ... X Vin --) Vk pertenece a F y
(x1, x2, ... ,xn) pertenece a Ci1 X Ci2 X ... X Cin
entonces Fj (x1, x2, ... ,xn) pertenece a Ck

3) Estos son los unicos miembros de los Ci

En este caso decimos que el conjunto V = Cl ((Gi, i=1,n) , F)

En el caso del TAD STACK (de naturales), V tiene como miembros al conjunto de los STACKs y a N. La definicion anterior podria aplicarse de dos formas:

i) Considerando G = ((), ()) y F contendria todas las funciones necesarias para generar los stacks (New, Push, Pop) y a los naturales (CERO, SUCC). En este caso estariamos definiendo en una misma algebra ambos tipos.

ii) Siguiendo el criterio discutido anteriormente, puede considerarse predefinido al conjunto N, y definir el algebra correspondiente a los STACKs con:

G = ((), N), F = (New, Push, Pop)

Aplicaremos el segundo criterio, por las razones ya expuestas, el cual podemos enunciar con generalidad de la siguiente forma:

En un algebra de tipo, el TOI se define como Cl (((),I), S), siendo I=V-TOI o sea aquellos dominios del algebra distintos del TOI y siendo S el conjunto de operaciones que en la especificacion sintactica aparecen con recorrido en el TOI es decir, de la forma:

Vi1 x Vi2 x ... x Vin ----> TOI.

Los demas conjuntos del algebra se consideran definidos.

A su vez, el conjunto de operaciones S puede particionarse en dos conjuntos:

S1: el conjunto de las operaciones de la forma:

f : TOI x Vi1 x Vi2 x ... x Vik -----> TOI. (Ej: SUCC, push, pop),que como se ve, requieren como argumento al menos un elemento del TOI.

Para que alguna de estas operaciones pueda ser aplicada, deberan existir funciones que generen elementos del TOI sin usar argumentos que pertenezcan a el. Estas forman:

S2: el conjunto de las operaciones de la forma:

f : Vi1 x Vi2 x ... x Viq ---> TOI, con Vih distinto al TOI,h= 1,..,q

Las operaciones del conjunto S2, en general, se aplican a n-uplas donde ningun componente pertenece al TOI. Un recurso de notacion se emplea para definir constantes del TOI (como el Cero). Estas se denominan funciones nularias, o sea sin argumentos y pertenecen, por lo tanto, al conjunto S2. Son ejemplos las funciones CERO y New.

2.2.2 EL AXIOMA DE INDUCCION

Nos interesa ahora introducir un axioma fundamental, asociado a la definicion de clausura, que es el axioma de induccion(L&Z77).

La definicion de naturales, que parcialmente introdujimos mas atras incluye el siguiente axioma:

Si dada una propiedad P se cumplen 1 y 2 tales que:

1) P vale para el natural cero
2) P vale para n implica que vale para SUCC(n),

entonces P vale para todos los naturales.

Esta idea puede generalizarse:

Si dada una propiedad P se cumplen 1 y 2 tales que:

1) P vale para todos los elementos del TOI generados por operaciones del conjunto *S2* (definido mas arriba)
2) P vale para t1, t2,...., tp, implica que vale para el elemento F(t1,t2, ...,tp,v1, ..., vq) para toda operacion F del conjunto *S1* (definido mas arriba), donde
   t1,...,tp pertenecen al TOI y
   v1,...,vq no pertenecen al TOI

entonces P vale para todos los elementos del TOI.

Este axioma esta implicito en la definicion por clausura de los TOI conduce a un metodo para probar propiedades de estos: la demostracion por induccion. Esto, que se hacia en cursos liceales con los naturales, aparece aqui generalizado.

Asi, para los *STACKS*, si queremos probar alguna propiedad P cualquiera basta:

1) Probar P(New( )) (Unica operacion del tipo *S2*)
2) *Suponer* P valida para un stack generico *S* y demostrar que entonces P vale para Push(s, h), (siendo h un natural cualquiera) y Pop (s).

2.2.3 PROPIEDADES DE LAS OPERACIONES *S*

Por ahora solo nos hemos referido, como puede verse, a una parte de la especificacion sintactica. No hemos hablado ni de operaciones cuyo recorrido sea distinto del TOI ni de la especificacion semantica del tipo.

De las primeras se hablara en la seccion siguiente. En este punto nos concentraremos en las ecuaciones de la especificacion semantica que corresponden a las operaciones del conjunto *S* (es decir, aquellas cuyos miembros son composiciones de funciones que dan como resultado elementos del TOI).

Veamos en particular el axioma:

$$Pop\ (Push(s, n)) = s$$

A partir del axioma se ve que si a un stack *S* se aplica Push con un natural cualquiera y, a continuacion, Pop, volvemos a obtener el mismo stack.

En algun sentido Push y Pop se definen como inversos entre si.

Este axioma induce una propiedad importante: de todas las composiciones posibles de Push, Pop y New hay algunas de ellas que dan el mismo resultado, por ejemplo: Pop(Push(New, n)) = New.

Estos axiomas definen relaciones de equivalencia entre los elementos del TOI. Este conjunto queda, entonces, particionado en clases de equivalencia segun aquella relacion.

Dos expresiones z, w pertenecen a la misma clase si existe una secuencia de igualdades (deducibles de los axiomas) tales que:
z = z1 = z2 = ... = zn = w.

Podemos asociar a cada clase un unico elemento (canonico).
En el ejemplo, se puede demostrar que todos los elementos canonicos pueden generarse por composiciones adecuadas de las funciones New y Push.
Como caso particular: Pop (Push (New, n)) y New pertenecen a una misma clase cuyo elemento canonico es el elemento New.

Este resultado es fundamental pues establece que incluido en *S* hay un subconjunto de operaciones suficiente para generar el TOI (llamados operadores constructores que designaremos con la letra C) y el complemento no genera valores que no puedan ser generados por los anteriores (se llaman extensiones, designados con la letra E).

En el STACK, New y Push seran constructores y Pop una extension. No vamos a incluir aqui la demostracion de que Pop es una extension.
Basta puntualizar que:

Dentro del conjunto *S*, la semantica puede establecer cierta "superposicion" en el efecto de las operaciones y que de acuerdo a esto puede definirse los constructores como el minimo conjunto de operaciones *S* que pueden generar todo el conjunto TOI.

2.3. LAS OPERACIONES CON RECORRIDO FUERA DEL TOI

Hasta ahora nos hemos concentrado en estudiar las operaciones cuyo recorrido es el conjunto TOI que estamos definiendo. Ahora, vamos a centrar nuestro estudio en las operaciones cuyo recorrido no es el conjunto TOI. Nombraremos O al conjunto de operaciones con recorrido fuera del TOI. Se vera como estas operaciones resultan imprescindibles para manipular con utilidad los objetos que se estan definiendo.

Retomemos la especificacion del TAD NATURALES:

```
        | CERO : -----> N
       (|
        | SUCC : N -----> N
```

Se pueden realizar algunas observaciones:

1) Son todas operaciones del conjunto S(segun fue visto).

2) No existe especificacion semantica.

3) La clausura de (CERO, SUCC) sobre el conjunto vacio define el TOI :

$$(CERO, SUCC (CERO), SUCC (SUCC (CERO)),...., SUCC^{n} (CERO),.....)$$

(donde $SUCC^{0}(CERO) = CERO$ y $SUCC^{n}(CERO) = SUCC^{n-1}(CERO)$).

A partir de las observaciones anteriores se pude concluir que:

i) No es posible probar que este conjunto sea el de los naturales definido por los axiomas de Peano.
En particular,por ejemplo,nada asegùra que CERO sea distinto de SUCC(CERO).
Si esto no ocurriera, todos los elementos del TOI serian iguales. La propiedad CERO <> SUCC (CERO) es asegurada por uno de los axiomas de Peano, el cual no fue incluido en la especificacion.

ii) Una operacion que permita determinar la igualdad/desigualdad de dos naturales deberia tener la siguiente sintaxis:
EQUAL : N X N -----> B (Siendo B un conjunto con dos elementos diferentes T y F que representan los dos resultados posibles de la funcion).
Esta funcion tiene recorrido en un conjunto diferente al TOI.
Es, por lo tanto, una operacion tipo O.

Se ve, con este pequenio ejemplo que al no existir operaciones tipo O no es posible asignarle propiedades que caracterice a los elementos del TOI. Vamos a modificar levemente nuestro tipo abstracto.

Vamos a definir:

```
CERO : -----> N
SUCC : N x N -----> N
EQUAL : N x N -----> BOOLEAN
```

1 EQUAL (CERO, CERO) = T
2 EQUAL (CERO, SUCC (n)) = F
3 EQUAL (SUCC (n), CERO) = F
4 EQUAL (SUCC (n1), SUCC (n2)) = EQUAL (n1, n2)

Los axiomas de Peano omitidos en la especificacion anterior aparecen aqui en los axiomas 2 al 4. El axioma de induccion, como ya se ha dicho, esta implicito en la especificacion.

Es importante destacar que ahora con la operacion EQUAL es posible distinguir los elementos del TOI. Se ve que es una operacion que OBSERVA (de alli O, el nombre del conjunto de estas operaciones) el comportamiento de los elementos del TOI cuando les son aplicadas operaciones del conjunto S.

Hasta ahora, dado un elemento del TOI, le aplicabamos una operacion de S y se obtenia un elemento del TOI.

Pero, al realizar esta operacon ?se obtiene un elemento distinto del original?

Solamente podemos verificar que dos elementos son distintos cuando tienen comportamientos diferentes frente a una misma operacion.

Por ejemplo, sean los naturales CERO y SUCC (CERO) (resultado de aplicar la operacion SUCC al natural cero dado). Para verificar que son distintos, usando el axioma 2, se ve que si n = CERO, EQUAL(CERO, SUCC (CERO))= F y por el axioma 1 EQUAL (CERO, CERO) = T.

En otras palabras, si al elemento CERO se le aplica la funcion SUCC, se obtiene un natural que tiene un comportamiento distinto que el CERO respecto de la funcion EQUAL. Entonces se ve que con las operaciones de O, se definen propiedades sobre los elementos del TOI. De aqui surge la conclusion siguiente:

LA SEMANTICA DEL TIPO ABSTRACTO ESTA DADA POR LA ESPECIFICACION SEMANTICA DE LAS OPERACIONES DE O.

Esta idea complementa el hecho que estamos definiendo el conjunto segun sea el comportamiento respecto a determinadas operaciones.

Con este ejemplo se recuerdan dos ideas fundamentales:

1) Se define el tipo abstracto indicando el comportamiento de sus elementos respecto a determinadas operaciones.
2) La semantica del TAD (el significado) esta dada por la especificacion semantica de las operaciones tipo O.

Un ultimo detalle a remarcar. Vimos que para que el tipo abstracto tuviera interes, es necesario que haya operaciones en O, o sea es necesario definir operaciones con recorrido en un conjunto diferente al TOI. Este conjunto debe estar definido, para lo cual creamos en general otro tipo abstracto de datos.

Por lo anterior se ve que, dado un TAD T cualquiera siempre existiran, en su especificacion, operaciones con recorrido en otro TAD T'.

Si tambien hubiera que definir el TAD T' se tendria nuevamente la misma situacion. Para evitar que la definicion de un TAD se transforme en una sucesion infinita de especificaciones, debe existir un TAD primitivo. Dicho TAD es BOOLEAN.

O sea, consideramos que EXISTEN DOS VALORES DIFERENTES (T y F).

3 ALGUNAS IDEAS SOBRE COMO CONSTRUIR ESPECIFICACIONES ALGEBRAICAS

3.0 OBJETIVO

Se definen propiedades que deben cumplir las especificaciones algebraicas. Luego se presenta informalmente un procedimiento para construir especificaciones algebraicas con dichas propiedades.

Es muy importante destacar el alcance de esta descripcion: NO pretende ser un tratado riguroso del tema, pues supera sobremanera los alcances del curso. Solo se pretende insinuar que hay una teoria matematica que respalda algunos resultados presentados que se aplicaran para llegar al objetivo de este estudio: tener una metodologia para disenar especificaciones algebraicas interesantes.

3.1 CONSISTENCIA Y COMPLETITUD

Vamos a definir el tipo "bolsa de enteros" con

NUEVA-BOLSA : ---> Bolsa
AGREGAR : Bolsa x entero ---> Bolsa
QUITAR : Bolsa x entero ---> Bolsa
PERTENECE : Bolsa x entero ---> BOOLEAN

1) PERTENECE (NUEVA-BOLSA,i) = F
2) PERTENECE (AGREGAR(b,i),j) = si i = j entonces T
sino PERTENECE (b,j)
3) QUITAR (NUEVA_BOLSA) = (NUEVA_BOLSA)
4) QUITAR (AGREGAR (b,i),j) = si i = j entonces b
sino AGREGAR (QUITAR (b,j),i)
5) PERTENECE (QUITAR (b,i),j) = si i = j entonces F
sino PERTENECE (b,j)

Consideramos el valor:
QUITAR(AGREGAR(AGREGAR(BOLSA_NUEVA,9),9),9) que pertenece al TOI (aplicando la clausura, ya vista anteriormente).
Este valor tiene la siguiente propiedad:

a) Usando el axioma 4) vemos que es igual a AGREGAR(BOLSA_NUEVA,9) y luego, usando el axioma 2) llego a:
PERTENECE(QUITAR(AGREGAR(AGREGAR(BOLSA_NUEVA,9),9),9),9) = T
b) Usando el axioma 5) llego a:
PERTENECE(QUITAR(AGREGAR(AGREGAR(BOLSA_NUEVA,9),9),9),9) = F

Entonces la funcion PERTENECE, aplicada al mismo elemento da como resultado dos valores diferentes lo cual contradice el hecho de que es una funcion. Llegamos a una contradiccion en los axiomas o, de otra manera, los axiomas son inconsistentes.
De aqui que una propiedad deseable es que la especificacion sea CONSISTENTE. Se puede ver que si eliminamos el axioma 5), la especificacion deja de ser inconsistente. De ahora en adelante trabajaremos solo con los axiomas 1) al 4).

Ahora, la especificacion algebraica debe indicar en los axiomas el comportamiento de las funciones para TODOS los elementos del conjunto. Por ejemplo, si NO usamos el axioma 1) (nuestra especificacion solo contiene los axiomas 2) al 4)) vamos a llegar a que NO esta definida la operacion PERTENECE para el elemento NUEVA_BOLSA, o sea que la especificion no es suficientemente completa.
Una especificacion es SUFICIENTEMENTE COMPLETA si todos los resultados posibles se pueden deducir de los axiomas.
(Esta es una nocion intuitiva, de un concepto que puded definirse con rigor).

Lo importante es recalcar que son conceptos diferentes. Por ejemplo:

1) Si tenemos los axiomas 1) al 4) la especificacion es CONSISTENTE Y SUFICIENTEMENTE COMPLETA.
2) Si consideramos los axiomas 1) al 5) la especificacion NO ES CONSISTENTE pero SI SUFICIENTEMENTE COMPLETA.
3) Si consideramos los axiomas 2) al 4) la especificacion es CONSISTENTE pero NO SUFICIENTEMENTE COMPLETA.
4) Si consideramos los axiomas 2) al 5) la especificacion NO CUMPLE NINGUNA DE LAS DOS CONDICIONES.

3.2 METODO HEURISTICO PARA CONSTRUIR ESPECIFICACIONES

Un problema interesante de resolver es el siguiente: dadas de un TAD cualquiera, las especificaciones sintacticas y semantica, dicha especificacion es consistente?, ?dicha especificacion es suficientemente completa? Podriamos interesarnos en condiciones NECESARIAS y SUFICIENTES (algun conjunto de condiciones tales que: se verifican dichas condiciones (===) la especificacion es consistente (o suficientemente completa)).

Desafortunadamente se puede probar que NO existe tal condicion.
Entonces se veran condiciones mas debiles, en particular, condiciones suficientes para la completitud suficiente de la especificacion.
Se vera un metodo que POR CONSTRUCCION garantiza que la especificacion creada sea suficientemente completa. De otra manera: si construimos la especificacion usando el metodo nos aseguramos que la especificacion cumple esta propiedad (condicion suficiente), pero pueden existir especificaciones suficientemente completas NO construidas a partir de este metodo. (Por eso, no es necesaria).
El metodo presentado es una simplificacion del algoritmo en (G&H78) (donde puede estudiarse con rigor).

Vamos a ver el metodo con un ejemplo, indicando desde ya que va a haber expresiones informales (no rigurosas), dado que un estudio completo escaparia de lejos el alcance del curso. Vamos a intentar construir una especificacion algebraica SUFICIENTEMENTE COMPLETA para el TAD bolsa de enteros:

NUEVA_BOLSA : ---> bolsa
AGREGAR : bolsa x entero ---> bolsa
QUITAR : bolsa x entero----> bolsa
PERTENECE : bolsa x entero ---> boolean

Nuestro objetivo es crear un conjunto de axiomas SUFICIENTEMENTE COMPLETO. Entonces:

1) PARTICIONAR LAS OPERACIONES EN LOS CONJUNTOS C,E y O

Mientras las operaciones de O son faciles de reconocer,no lo es tanto dividir el conjunto S en los subconjuntos C y E. Por ejemplo: podriamos considerar como constructores (C) a (NUEVA-BOLSA,AGREGAR, QUITAR), y no tenemos ninguna operacion tipo E. Pero tambien podriamos definir C = (NUEVA-BOLSA, AGREGAR) E = (QUITAR) o C=(NUEVA-BOLSA, QUITAR), E = (AGREGAR).
En el caso general no es sencillo saber cuales operaciones son Constructores y cuales son Extensiones. En la practica, nosotros tenemos alguna nocion de las propiedades del conjunto (incluso los nombres de las operaciones son mnemotecnicas: AGREGAR, QUITAR nos indican alguna idea), con la cual podemos decir cuales son constructores y cuales no. Este es el caso de los ejemplos que veremos nosotros.
Vamos a considerar entonces que las operaciones se dividen:

```
┌──────────────────────────────┐
| C = (NUEVA-BOLSA, AGREGAR)   |
| E = (QUITAR)                 |
| O = (PERTENECE)              |
└──────────────────────────────┘
```

Es importante destacar que las operaciones nularias pertenecen al conjunto C.

2) CONSTRUIR EL CONJUNTO CTERMS = (c(x1, ...,xn) donde c es una funcion que pertenece a C y para todo i, xi es una variable independiente del tipo apropiado segun la sintaxis de C).

En nuestro caso CTERMS = (NUEVA-BOLSA, AGREGAR (b,i)) donde b es un variable tipo bolsa e i una variable entera.

3) CONSTRUIR EL CONJUNTO OTERMS = (o(x1,...,xn) donde o es una funcion que pertenece a O , si xi no pertenece al TOI es una variable independiente y si xi pertenece al TOI es un elemento de CTERMS).

En nuestro caso tenemos:
OTERM = (PERTENECE? (NUEVA-BOLSA, j),PERTENECE?(AGREGAR(b, i), j))

Vamos a ver mas detenidamente estos conjuntos. CTERMS seria el conjunto de todas las funciones de C aplicadas a cualquier variable.
Intuitivamente se ve que si a las variables (en este caso i, b) le asignamos todos los valores posibles obtenemos todos los elementos del TOI (de alli que a las funciones de C se les llame constructores).
OTERMS es el conjunto de operaciones de O aplicadas a todos los elementos de CTERMS, o sea a TODOS los elementos del conjunto TOI.
En nuestro caso si damos a j cualquier valor entero, como con _BOL-NUEVA_BOLSA y AGREGAR (b, i) se obtienen todos los elementos del TOI, obtenemos todos los casos posibles a los cuales se puede aplicar la funcion PERTENECE. Se garantiza asi la completitud de la misma.

4) CONSTRUIR EL CONJUNTO ETERMS= (e(x1, ..., xn) e es una funcion que pertenece a E , si xi no pertenece al TOI es una variable independiente y si xi pertenece al TOI es un elemento de CTERM).

En nuestro caso :
ETERMS = (QUITAR (NUEVA_BOLSA), QUITAR(AGREGAR(b, i), j)).
Es la misma idea del conjunto OTERMS pero para las funciones de E.

5) CONSTRUIR LOS AXIOMAS, tomando como parte izquierda todos los elementos del conjunto OTERMS, y como partes derechas los resultados de aplicar las funciones correspondientes.

En nuestro caso, vamos a tener dos axiomas :

1) PERTENECE (NUEVA_BOLSA, j) = ...
2) PERTENECE (AGREGAR(b, i), j) = ...

donde cada parte izquierda en un elemento de OTERMS. Aqui le damos significado a las operaciones tipo O(realizamos la semantica de dichas operaciones).

La idea es que las partes derechas sean mas simples que las izquierdas. En nuestro caso :

PERTENECE (NUEVA_BOLSA, j) = F
PERTENECE (AGREGAR (b, i), j) = IF i=j THEN T
ELSE PERTENECE(b,j)

Notar que las partes derechas son en cierto sentido mas sencillas que las izquierdas, en el hecho de que son constantes o la funcion PERTENECE (b, j) no aparece compuesta con la funcion AGREGAR.

Estas ideas pueden estudiarse con rigor en (G&H78).

6) COMPLETAR LA AXIOMATIZACION usando las equivalencias de las funciones de la clase E en la clase C; todas las partes izquierdas son los elementos de ETERMS

En nuestro caso tendremos : QUITAR (NUEVA_BOLSA,j) = ...
QUITAR (AGREGAR (b,i),j) = ...

Entonces completamos las equivalencias

c) QUITAR(NUEVA_BOLSA,j) = NUEVA_BOLSA
d) QUITAR (AGREGAR (b,i),j) = si

i = j entonces b

sino AGREGAR (QUITAR(b,j),i),

La verificacion de la consistencia tiene reglas que derivan de la logica.Un ejemplo de ellos es la regla de Church-Posser pero NO profundizaremos mayormente en el tema. En los casos en que trabajamos nosotros la consistencia es practicamente trivial,dado que si se entendio el problema y las operaciones a realizar,la consistencia va a estar casi siempre garantida.

4. APENDICE.-

Es importante destacar que para realizar las especificaciones solo precisamos 5 primitivas (o sea herramientas que suponemos que existen sin definicion y son utilizadas), a saber:

a)la composicion de funciones (QUITAR(AGREGAR(b,i),j))

b)la relacion de igualdad (=)

c)TRUE |
)---)constantes que asumimos que son DISTINTAS
d)FALSE| (como vimos al final de 2)

e)cantidad ilimitable de variables (b,i,j)

Se puede ver que SI_ENTONCES_SINO puede especificarse:

SI_ENTONCES_SINO : BOOLEAN X T X T ----) T

1) SI_ENTONCES_SINO (TRUE, t1, t2) = t1
2) SI_ENTONCES_SINO (FALSE, t1,t2) = t2

SI_ENTONCES_SINO (b, t1, t2) aparece escrito en notacion infija:
SI b ENTONCES t1 SINO t2.

5. BIBLIOGRAFIA.-

G&H78. Guttag J., Horning J.
The Algebraic Specification of Abstract Data Types.
(Acta Informatica 10, 1978)

L&Z77. Liskov B., Zilles S.
An Introduction to Formal Specifications of Data Abstractions.
(en Current Trends in Programming Methodology, vol 1,
Prentice Hall, 1977)

SOBRE LA FORMACION UNIVERSITARIA EN INFORMATICA

Intento de síntesis de diversas reflexiones

Víctor Manuel Toro C.
Mario Castillo H.
Universidad de los Andes
Bogotá - *Colombia*

I. ANTECEDENTES

Los Congresos Latinoamericanos de Informática que anualmente organiza el CLEI (Centro Latinoamericano de Estudios en Informática) han sido un excelente foro para presentar y conocer los avances en Informática de los diferentes países del área. Pero más allá de este objetivo, han sido una ocasión privilegiada para el encuentro e intercambio entre profesores e investigadores de universidades latinoamericanas. Como es apenas natural, uno de los principales temas de discusión que surge es el de la formación universitaria en Informática.

Durante el XI Congreso, que tuvo lugar en Porto Alegre - Brasil - en julio de 1985, se reunió un grupo de trabajo para discutir más en detalle sobre este tema de la formación universitaria en informática: currícula, organización, objetivos, estructura, recursos, deficiencias, etc.. Este grupo, conformado por profesores universitarios de Argentina, Bolivia, Brasil, Chile, Colombia, Ecuador, Perú, Uruguay y Venezuela, se reunió a diario durante los seis días del Congreso.

Este documento es un intento de síntesis de las opiniones, reflexiones y experiencias discutidas al interior de dicho grupo. Sinembargo, obviamente no se pretende que los comentarios aquí formulados sean aplicables por igual a los diferentes países o universidades. En efecto, las universidades latinoamericanas cubren un amplísimo espectro, que abarca desde aquellas que tienen programas de doctorado de excelente nivel y adelantan investigaciones y desarrollos de punta, hasta otras que apenas están iniciando programas de formación en esta área y cuentan con muy limitados recursos humanos y materiales.

II. INTRODUCCION

La importancia que está tomando la Informática en la sociedad contemporánea es cada vez mayor, y esto exige de todos los estamentos de la sociedad un compromiso cada vez más directo. El papel que dentro de este proceso juegan diferentes estamentos de la sociedad (Gobierno, Estado, Industria, Universidad,...) es actualmente tema de debates y ajustes. Sin duda en la mayor parte de los países latinoamericanos continuan las polémicas sobre el papel que corresponde al Gobierno, al Estado y a la Industria Nacional. Por su parte, el papel de la Universidad se perfila tal vez con mayor claridad.

A la Universidad corresponde:

* La formación de recursos humanos con un alto nivel académico y técnico, conscientes de la realidad de su país y comprometidos laboral y socialmente con el desarrollo del sector.
* La *apropiación* (i.e. conocimiento + experiencia + sentido crítico) de los desarrollos de vanguardia proveniente de países industrializados que estén especialmente avanzados en el área.
* La generación de conocimiento a través de la búsqueda y el encuentro de soluciones apropiadas a su medio y recursos.

* La multiplicación e irradiación de este conocimiento hacia la la sociedad.
* La promoción del cambio y la innovación en el área.
* La colaboración con el Gobierno y con la industria a través de planes de desarrollo concertados.

El objetivo de este trabajo es entonces intentar un balance realista y autocrítico del estado actual de la Informática en nuestras Universidades bajo la óptica de los objetivos anteriormente planteados. Se han distinguido entonces cuatro aspectos:

* La formación de profesionales en Informática.
* La formación y apropiación de la Informática por la comunidad universitaria en general.
* La utilización de la Informática como apoyo al proceso mismo de enseñanza-aprendizaje.
* El papel del Estado en el proceso de desarrollo de la Informática en las Universidades (públicas y privadas).

III. FORMACION PROFESIONAL EN INFORMATICA

El primer punto que se analiza es el de la formación de profesionales en Informática, aspecto crucial dado que son justamente estos profesionales quienes se constituyen en los actores y motores fundamentales en el desarrollo Informático de un país.

Los aspectos curriculares suelen ser los primeros que se evocan al plantear este tema de la formación profesional. Son varios los aportes importantes que existen en materia de diseño y estructuración curricular en Informática: las principales sociedades profesionales (ACM, IEEE, IFIP, AFCET,...) presentan cada cierto tiempo sus recomendaciones en esta materia; más recientemente, la

Unesco en asocio con la IFIP elaboraron un Currículo Modular en Informática [**], el cual se ha constituído en uno de los trabajos más completos en este aspecto; finalmente, varias universidades latinoamericanas que han llegado a una cierta madurez en el área han publicado documentos en los que presentan detalladamente su curriculo, incluyendo no solo los contenidos, sino también objetivos, metodología y organización [***].

Dado que el aspecto curricular ha sido en general bastante discutido y documentado, el grupo de trabajo de Porto Alegre consideró entonces más relevante concentrar la atención en la conformación y organización de las unidades docentes-investigativas encargadas de impartir la formación y de promover la apropiación de la Informática en nuestras Universidades. En otras palabras, en lugar de entrar a preguntarnos qué se enseña en Informática, en qué orden, con qué método,..., se ha cuestionado quién enseña la Informática, con qué recursos cuenta, cuál es la estructura organizativa, cuáles son los objetivos y la filosofía de esta enseñanza y qué política institucional los rige.

A continuación se discuten someramente algunos aspectos:

- Recursos Humanos (profesores e investigadores) con los que cuentan dichas unidades.
- Facilidades de equipos y laboratorios de computación.
- El concepto de diseño y desarrollo curricular

[**] : "A Modular Curriculum in Computer Science"
Unesco - IFIP
7, Place de Fontenoy; 75700 Paris
ISBN: 92-3-102154-4(ed. inglés) o 92-3-302154-8(ed. español)

[***]: Univ. Aut'ooma de Mexico, Univ. Sim'o Bolivar - Caracas, Univ. de los Andes - Bogotá, Pontificia Universidad Católica de Rio de Janeiro, Univ. de Porto Alegre (Brasil), Univ. Católica de Santiago, etc.

III.1 Recursos Humanos

La consolidación de un programa universitario en Informática requiere ineludiblemente la conformación de un núcleo sólido y estable de profesores de planta, es decir, profesionales de informática (preferiblemente con nivel postgrado) , cuya (casi)totalidad de capacidad de trabajo esté dedicada al hacer académico. Aunque de hecho la anterior afirmación es evidente, muchas universidades en nuestros países funcionan en torno a un muy reducido grupo de profesores de planta (en general encargados de labores administrativas), y un muy alto porcentaje del trabajo docente es hecho por profesores de "hora-cátedra".

De hecho, las Universidades suelen tener dificultades grandes para poder conformar un grupo estable de profesores de planta. Toda universidad aspira a integrar su cuerpo profesoral con profesionales particularmente destacados tanto en lo académico como en el desempeño profesional. Sinembargo, profesionales de esas características son también altamente demandados por las empresas. Esta competencia entre Empresa y Universidad muchas veces se resuelve en detrimento de esta última. En efecto, las limitaciónes presupuestales que suelen tener las Universidades dificulta muchas veces la contratación y en la mayoría de los casos la permanencia a largo plazo de profesores calificados y de experiencia en la universidad. Si bien esta situación se puede presentar en otras áreas, en la Informática se convierte en un problema realmente crítico dadas las atractivas ofertas de las empresas y la rápida obsolecencia académica de los conocimientos de un profesional en Informática que no esté inmerso en un ambiente de estudio.

No se puede hablar tampoco de un perfil profesional para el profesorado de Informática. Van desde ingenieros no informáticos que por su experiencia profesional en el área se han convertido en profesores, hasta jóvenes doctorados con poca o ninguna experiencia profesional pero con una muy actualizada formación académica. Son realmente muy pocos los profesores postgraduados en

Informática, con experiencia práctica en el área, y que lleven varios años de trabajo continuado en la universidad.

Otro punto que vale la pena destacar en este aspecto de los recursos humanos para la formación en Informática es el referente a la carga de trabajo sobre los profesores que integran las unidades responsables de esta formación. En efecto, en los últimos años se ha venido presentando en las Universidades una alta demanda por servicios docentes en Informática. Esta proviene, por un lado, del crecido número de bachilleres que aspiran a seguir esta carrera profesional, y por otro, de la creciente demanda de cursos en el área solicitados por otras Facultades o Departamentos. Las Universidades, particularmente las privadas, se han visto obligadas a dar cabida a esta demanda sobrepasando muchas veces sus capacidades reales, debido en general a razones financieras o de presencia institucional. Como resultado natural de dicha situación, las carreras de Informática suelen ser los programas profesionales más grandes de las Universidades que otorgan título profesional en el área.

Lo anterior ha conducido casi sin excepción a una notoria sobrecarga de trabajo docente y administrativo en las unidades de Informática. Los profesores de planta han debido aumentar considerablemente su carga de trabajo, llegando en algunos casos extremos hasta unas 20 horas de clase por semana, a dirigir un volumen muy elevado de proyectos de grado, y a asumir no pocas labores administrativas. Así mismo, muchas Universidades han recurrido al empleo intensivo de profesores de hora cátedra, sin tener en todos los casos buenas posibilidades para su selección.

En general esta deformación de la labor profesoral, que reduce el trabajo académico al simple dictar clases y a las labores administrativas adjuntas, conlleva a una rápida decepción del profesor con respecto a sus posibilidades de desarrollo profesional en el ambiente universitario. Es evidente que esta situación en muchos casos no solo ha deteriorado la calidad de la docencia, sino que se ha constituído en un serio obstáculo al surgimiento

de un ambiente académico apropiado para el desarrollo profesoral, y por ende de la Informática en nuestras universidades.

III.2 Equipos y Laboratorios de Computación

Una de las limitaciones más comunes en muchas de nuestras Universidades es el déficit de equipos de computación, tanto de microcomputadores como de computadores más grandes, en los cuales los estudiantes puedan adelantar las prácticas que sus cursos requieren. De hecho, no son pocas las universidades latinoamericanas en las que los estudiantes solo cuentan con unas pocas horas al semestre para efectuar algunas prácticas superficiales, a veces sobre equipos obsoletos que desde hace varios años se encuentran fuera del mercado.

Son varias las razones que generan este problema de limitación de equipos computacionales:

- Insuficiencia de recursos económicos.
- Excesiva tramitación burocrática para las decisiones de adquisición de equipo. Este problema es particularmente grave en las universidades estatales.
- Limitaciones del régimen de importaciones y de aranceles de varios de los países del área, que no contempla verdaderas facilidades o prioridades para la importación de equipos computacionales con fines académicos.
- Ausencia de una política real de apoyo al desarrollo universitario por parte de los distribuidores o fabricantes de equipos.
- Individualismo e incapacidad de asociación efectiva de las Universidades para enfrentarse en grupo a los obstáculos anteriores.

La afirmación anterior sobre la frecuente 'obsolesencia' de los equipos en que los estudiantes realizan sus prácticas tiene un sentido bien definido. No se trata obviamente de una imposición de la "moda", según la cual el ideal sea tener acceso a los últimos productos, aditamentos o versiones ("gadjets"): De hecho la muy rápida evolución del mercado hace imposible que nuestras universidades pretendan mantenerse sintonizadas con los últimos desarrollos. No es tampoco la concepción de que un estudiante deba efectuar sus prácticas en el mismo tipo de equipos que podría encontrar en las empresas: Es evidente que la misión de la Universidad no es enseñar a manejar un determinado tipo de equipos o productos.

El verdadero sentido de la 'obsolecencia tecnológica' es el aspecto de costos. A pesar de que en muchos casos equipos de finales de los 70's pueden ofrecer un espacio relativamente apropiado para prácticas e investigaciones en el contexto universitario, el mantenimiento y soporte de estos suele ser muy costoso. Así mismo, la relación *'performance'/costo* de dichos equipos es muy baja comparada con la que tecnologías recientes ofrecen.

Es entonces frecuente el caso de universidades que hoy están amarradas a grandes equipos que adquirieron hace 8, 10 ó más años, el cual representó un gran esfuerzo y avance en su momento. Y sinembargo, en la actualidad estos equipos se van constituyendo en un lastre financiero, administrativo y operativo para el desarrollo Informático de nuestras Universidades.

Para concluir este punto vale la pena mencionar un criterio que impulsa una comisión mixta de la ACM y la IEEE para evaluar los programas en ciencias de computación, según el cual cada estudiante debería tener acceso garantizado al computador por al menos una hora diaria por cada curso relacionado con Informática que esté tomando [**].

[**]: "ACM - IEEE accreditation criteria for programs in Computer Science"
IEEE - The Institute, january 1985

III.3 El concepto de Diseño y Desarrollo Curricular

Es un hecho que la Informática es una ciencia nueva, que se está renovando muy rápidamente, ante la cual existen grandes expectativas y sobre cuya enseñanza e investigación se posee una experiencia no muy amplia en nuestros países. Más aún, la importancia cada vez mayor que esta disciplina está adquiriendo en casi todos los sectores de actividad van poco a poco haciendo de la Informática una "Ciencia Básica". Esta situación contrasta, por ejemplo, con ciencias básicas como matemáticas o física, en las que se posee una tradición y experiencia docente y de investigación mucho más amplia, y en las cuales el aspecto de Metodología de la enseñanza y de la investigación ha sido largamente estudiado.

En consecuencia, configurar un currículo de estudios profesionales en el área es una tarea difícil y compleja, no solo en la determinación de los aspectos de contenido, sino también y principalmente en aspectos de metodologías, recursos (humanos y materiales), criterios de evaluación y ajuste, etc.

Al recopilar las experiencias de diseño y desarrollo curricular expuestas por varios de los profesores del grupo de trabajo de Porto Alegre, pareciera deducirse que ha habido un trabajo bastante intensivo y detallado en torno a la estructuración del currículo de la carrera profesional de Informática. Inclusive, se podría afirmar que en no pocas ocasiones este trabajo llega hasta niveles realmente muy minuciosos en cuanto a contenidos, requisitos, bibliografía y objetivos específicos de cada curso, seminario o laboratorio. Más aún, algunas veces este trabajo de renovación curricular se ve obtaculizado y dilatado por un excesivo purismo metodológico que llega hasta opacar el objetivo académico inicial, convirtiendo las discusiones metodológicas en parte del problema y no en parte de la solución.

Obviamente este trabajo detallado de diseño y estructuración curricular no tiene 'per se' ninguna connotación negativa. El problema suele ser que esta focalización en torno al *currículo del*

estudiante ocasiona que a veces se descuide otro aspecto fundamental para el fortalecimiento de un programa académico: el planteamiento de *políticas y mecanismos para el desarrollo curricular de los profesores.* Por esto último se entiende:

- el desarrollo, la calidad y enriquecimiento intelectual del cuerpo profesoral.
- el fortalecimiento de la organización del trabajo docente e investigativo de los profesores.
- la infraestructura necesaria de equipos y laboratorios que permita soportar las necesidades generadas por el currículo y facilite a los profesores la utilización de dichos recursos con una cierta comodidad.

IV. COMUNIDAD UNIVERSITARIA E INFORMATICA

Hasta hace unos pocos años la Informática era dominio exclusivo de los especialistas y hoy, independientemente de la opinión y del estado de desarrollo de nuestras Universidades al respecto, ésta se ha convertido en un elemento básico de la cultura contemporánea. Mientras muchas veces las universidades prolongan discusiones sobre la importancia y conveniencia del desarrollo de la Informática en la comunidad universitaria en general, importantes sectores de nuestra sociedad se han visto sacudidos, cuando no atropellados, por un proceso de informatización acelerado y no pocas veces traumático.

Mientras hay universidades que siguen buscando una política de desarrollo Informático clara, coherente, segura y unificada antes de tomar sus decisiones, muchas universidades de los países industrializados e inclusive de algunos países latinoamericanos se han decidido por actuar de manera rápida, afrontando los riesgos que sus determinaciones conllevan. En los países latinoamericanos

la Informática es de hecho un sector estratégico, económica y socialmente, y como tal requiere un tratamiento especialmente atento y ágil en nuestras Universidades.

Se presentan a continuación algunas reflexiones sobre el estado actual la Informática en la comunidad universitaria en general (es decir, en una dimensión global, y no únicamente limitada a la formación profesional en Informática):

- Estructura y Organización de la docencia en Informática
- Desarrollo de la Informática en las diferentes profesiones

IV.1 Estructura y Organización

Se pueden distinguir cuatro formas típicas de organización de los servicios docentes en Informática:

* En las Universidades que poseen un programa profesional en Informática existe en general una unidad administrativa (Programa, Departamento ó Facultad) que concentra los profesores del área, la cual además de atender a los estudiantes propios de la carrera, coordina, asesora y realiza los cursos de formación en Informática que requieran las demás áreas de la Universidad.

En aquellas Universidades que no cuentan con un programa profesional en Informática se encontraron los siguientes esquemas:

* Existe una unidad docente de servicio, la cual eventualmente concentra a los profesores del área, responsable de coordinar y realizar los cursos de formación en Informática que requieren los diversos programas de la Universidad.

* Una unidad típicamente administrativa (generalmente el Centro de Cómputo), se ha visto avocada a desempeñar la función de unidad docente, con responsabilidades similares a las mencionadas en el párrafo anterior.

* Los cursos de Informática son concebidos, diseñados, dictados y evaluados por los diversos programas profesionales en forma independiente.

En general los dos últimos esquemas presentan las mayores debilidades en cuanto a la penetración de la Informática en las diferentes disciplinas. En efecto, con frecuencia dichos esquemas dan lugar a problemas como:

- dispersión e ineficiencia en la utilización de los recursos profesorales.
- deterioro del nivel académico de los cursos.
- falta de una coordinación adecuada, que se refleja en duplicación de cursos elementales y carencia de cursos más avanzados.
- esta dispersión de recursos y esfuerzos académicos ocasiona una falta de presencia del sector Informático de la Universidad con respecto a las otras áreas.

Parece evidente que es conveniente una centralización de los servicios docentes de Informática. Obviamente dicha centralización no debe ir acompañada de ningún tipo de imposición de contenidos académicos o monopolio de recursos. Esta tiene sentido únicamente en la medida en que, teniendo en cuenta las necesidades particulares de cada profesión, permita capitalizar los avances técnicos y metodológicos, y garantizar un nivel académico elevado y uniforme.

IV.2 Desarrollo de la Informática en los diferentes programas profesionales

Es clara la tendencia que existe en muchas carreras profesionales, desde las más cercanas a las artes y a las humanidades hasta las relacionadas directamente con la tecnología, hacia la introducción de materias básicas de Informática como parte de sus

currícula de estudios. Sinembargo, esta intención se ha visto en muchos casos limitada por insuficiencia de recursos profesorales y de equipos computacionales.

Es importante anotar que en la mayoría de los casos la formación Informática impartida a los estudiantes de otras carreras se ha limitado a algunos cursos de introducción a la programación (no siempre con la metodología y el lenguaje apropiado !), cuya utilidad práctica en la respectiva carrera es altamente cuestionable. Más que aprender a programar lo importante en la mayoría de las profesiones es adquirir una cultura y una experiencia de usuario consciente de las posibilidades y espacios que abre la Informática como herramienta de trabajo profesional.

El avance Informático de una carrera profesional no es necesariamente proporcional al número de cursos de Informática que hagan parte de su currículo, ni al número de equipos de computación de que dispongan. El verdadero desarrollo lo dá el *nivel de apropiación* (i.e. conocimiento + experiencia + capacidad crítica) por parte de los estudiantes de las herramientas informáticas aptas para la aplicación en su respectiva profesión.

Es claro que este último objetivo es el más dificil de lograr, puesto que exige a su vez un inmenso esfuerzo por parte de los profesores de la profesión respectiva tendiente a lograr dicha apropiación para ellos mismos con mucha mayor profundidad, de manera que puedan posteriormente trasmitirla a sus estudiantes.

V. APOYO DE LA INFORMATICA AL PROCESO ENSENANZA-APRENDIZAJE

Una de las realidades más preocupantes que se hizo patente durante las reuniones de Porto-Alegre fué el muy pobre nivel de utilización de la Informática como apoyo al proceso de Enseñanza-Aprendizaje propiamente dicho. Son muy pocos los recursos con que cuentan nuestras universidades en materia de:

- software especializado para la enseñanza de materias específicas.
- bancos automatizados de ejercicios y problemas disponibles para profesores y estudiantes.
- software de simulación que sirva de apoyo a las prácticas de laboratorio.
- sistemas autores para el desarrollo de ambientes de enseñanza.

Pero sinembargo, talvez la limitación más evidente es la escasez de equipos de computación disponibles para que los estudiantes realicen sus sesiones de aprendizaje apoyado por computador. En efecto, en general los equipos que exiten están preferencialmente destinados o bien a las prácticas de los cursos de Informática, o bien al uso de aplicaciones estándares (bases de datos, editores de texto, paquetes estadísticos, hojas electrónicas, cálculos numéricos,...).

Los pocos desarrollos que se reportaron en esta área han sido realizadas por pequeños grupos de profesores que comparten su interés en el tema, pero cuyas realizaciones todavía no han alcanzado a permear hasta los estudiantes. Como ya se mencionó, las razones principales de esta situación son la insuficiencia de equipos y la falta de productos terminados apropiados a nuestras necesidades.

VI. EL PAPEL DEL ESTADO EN EL DESARROLLO DE LA INFORMATICA EN LAS UNIVERSIDADES

Cabe preguntarse en qué medida los organismos del Estado que tienen bajo su tutela y control el desarrollo y el apoyo a la formación universitaria han contribuído a la situación descrita en los puntos anteriores. Es frecuente la opinión de que su papel se ha limitado fundamentalmente al aspecto normativo y de fiscalización, siendo en general reducida la *actividad de fomento* y apoyo que de hecho también deberían desempeñar.

En muchos casos el papel del Estado se ha limitado en buena parte a la aprobación de programas universitarios, y a la renovación periódica de los permisos de funcionamiento. Para esto se suele exigir la elaboración de un detallado informe escrito y se realizan algunas visitas de inspección.

La evaluación debería ser más exigente en aspectos mucho más fundamentales que van más allá del mero programa curricular que deberán seguir los estudiantes. Se deberían examinar en detalle aspectos cruciales como los recursos profesorales, los equipos computacionales y de laboratorio, la calidad y variedad de la biblioteca y documentación, etc., que la Universidad que presenta el programa posee o planea poseer.

En la práctica rara vez se realiza un seguimiento cuidadoso y un *compromiso compartido* de los planes de desarrollo de las Universidades con el objetivo de garantizar el cumplimiento de los compromisos adquiridos con la sociedad al iniciar o al mantener en funcionamiento programas de formación en Informática.

En casi todos de los países latinoaméricanpos existen organismos especializados para el fomento y el apoyo tanto financiero como tecnologico y de mercadeo a sectores que se consideran claves para el país (sector cafetero, agrícola, minero, de agroindustria,...). Sinembargo, para el sector universitario no siempre

existen organismos reales de fomento, que puedan ser socios de los planes de desarrollo de las universidades.

Sin duda alguna las universidades se verían fortalecidas y mejorarían significativamente su calidad si las entidades estatales que fueron concebidas para el control y el fomento de la Educación Superior adquirieran un compromiso decidido de apoyo, asesoría y colaboración con el desarrollo de aquellos programas que estas mismas instituciones aprueban.

VII. A TITULO DE CONCLUSIONES

El desarrollo de la Informática en una universidad es un proceso largo que demanda sin duda mucho trabajo y entusiasmo. En las primeras etapas probablemente buena parte de los problemas y de las soluciones sean hasta cierto punto *cuantitativas*: aumentar la capacidad computacional, ampliar el cuerpo profesoral, extender el uso de la Informática a varios sectores de la universidad.... Pero al cabo de algún tiempo se llega a una situación donde los problemas y los retos son fundamentalmente *cualitativos*.

Una de las claves del desarrollo Informático será entonces el surgimiento y el fortalecimiento de un *Ambiente Académico* apropiado al desarrollo profesoral y estudiantil, pero sobre todo profesoral, pues son los profesores, componente trascendente del proceso educativo, quienes deben asegurar la dinámica de la renovación y el desarrollo académico.

El 'ambiente académico propicio al desarrollo profesoral' es una interacción compleja de varios factores, entre los cuales se destacan los siguientes:

- contratación de profesores calificados,
- incentivos a la labor profesoral:
  - . salarios razonables,
  - . reconocimiento al prestigio intelectual,
  - . libertad y confianza en los profesores,
- disponibilidad de tiempo efectivo para realizar investigación y actividades de desarrollo profesional dentro de la Universidad,
- dotación de equipos computacionales apropiados,
- agilidad y ordenamiento administrativo,
- apoyo a esquemas de trabajo en grupo,
- acceso a información (buenos recursos de biblioteca y documentación, congresos y eventos).

Obviamente, no nos referimos a una Universidad "descendida del cielo" donde los puntos antes mencionados han alcanzado su plena expresión. Se trata ante todo de un propósito común por el que deben trabajar todos los estamentos de la Universidad: Directivos, Profesores y Estudiantes.

La única alternativa decorosa es creer firmemente en que un ambiente académico como el antes descrito es posible en nuestras Universidades. La única actitud honesta es trabajar incansablemente para lograrlo. Solo bajo esta perspectiva la labor académica logrará su verdadera dimensión y la Universidad jugará el papel que realmente le corresponde en nuestras sociedades.

COMUNICACIONES

RELACION ENTRE ESTILO Y PORTABILIDAD EN LOS LENGUAJES DE PROGRAMACION

Juan Alvarez Rubio
Universidad de Chile
Santiago - *Chile*

RESUMEN

Los temas de Estilo y de Portabilidad han tenido tradicionalmente un tratamiento separado e independiente en la literatura acerca de los lenguajes de programación. La práctica demuestra sin embargo que en el desarrollo de software los dos aspectos deben considerarse simultáneamente definiendo un compromiso entre ambos.

En lo conciernente a Estilo de programación, se pueden distinguir al menos dos hitos importantes. El primero ocurre a fines de la década del sesenta, se inicia con la aparición del artículo de Dijkstra "The Go To Considered Harmful" y se consolida como la metodología de Programación Estructurada. A mediados de la década del setenta, la aparición del histórico trabajo "The Elements of Programming Style", de Kernighan y Plauger, demuestra que además de la Programación Estructurada es necesario considerar otros elementos para configurar un buen estilo.

Con respecto a Portabilidad, la preocupación se refleja en el desarrollo y la aprobación de los estándares de los lenguajes de programación. FORTRAN es el primer lenguaje que se normaliza. Casi una década después de su aparición se aprueba el estándar de 1966. Razones prácticas inducen a definir el PFORT (Portable FORTran) como un subconjunto portable. Una segunda norma se aprueba en 1978 (FORTRAN-77), y actualmente se está trabajando en FORTRAN-8x. COBOL aparece en 1959, y es el primer lenguaje que ha completado tres estandarizaciones: 1968, 1974 y 1985. Por su parte BASIC, diseñado en 1964, aprueba un estándar de un subconjunto mínimo recién en 1978 y una segunda versión se espera para los próximos años. Por su parte, Pascal y Ada tienen ya una estandarización y se está completando la primera norma de C.

La relación entre Estilo y Portabilidad es de orden inverso. La mayor portabilidad se logra limitándose a usar los subconjuntos mínimos de los estándares, proscribiendo elementos e instrucciones que mejoran cualitativamente el estilo. Analizando la relación al menos en los casos de COBOL, FORTRAN y BASIC, se ilustran las decisiones que en la práctica deben tomarse para balancear un buen Estilo con una Portabilidad razonable.

EXPLORACION DE TRES METODOS DE LA ENSEÑANZA DEL LENGUAJE LOGO

Investigadora: *Ruth Donoso Villegas*
Inv. ayudantes: *Luis Osorio Olivares, Soledad Silva Vidal, Eduardo Solís Troncoso*
Prof. observantes: *Matilde Chacón A., Alicia Frank H., Andrés Opazo, Luis Varas, Luis Letelier N., Mario Velásquez M.*
Pontificia Universidad Católica de Chile
Santiago - Chile

Reflexionando sobre conceptos vertidos acerca de la enseñanza del lenguaje LOGO, y teniendo presente comentarios surgidos del propio Seymour Papert como: "Pienso que sabemos hacer funcionar el LOGO, sin embargo estoy seguro que vamos a tener que aprender mucho acerca de como usarlo, desarrollarlo e integrarlo a los ambientes de aprendizaje de los niveles más avanzados" y otros. Hemos querido desarrollar un estudio exploratorio para de él, sacar fundamentos e hipótesis para otras investigaciones.

La experiencia se desarrolló con 180 niños de tres colegios diferentes, en cada colegio se eligío una muestra de 60 niños a los que enseñamos por tres métodos diferentes, el lenguaje LOGO.

Uno de los métodos utilizados es el mismo utilizado por Pea y Kurland del Bank Street College Center for Children and Technology, otro es el más utilizado en los colegios chilenos y el tercero es una adaptación basada en la teoría de Zoltan Dienes para el aprendizaje de las Matemáticas.

Los objetivos del estudio son:

- Determinar que métodos de enseñanza de lenguaje LOGO produce mayor aceptabilidad en niños de 10 a 14 años.
- Evaluar bajo cual método aplicado, el niño demuestra mayor creatividad.
- Diseñar un método de la enseñanza del lenguaje LOGO, donde el niño sienta placer de explorar y aprender.

Los resultados obtenidos de este estudio son muy interesantes y valederos para nuestros propósitos, ya que mostraron similitudes y diferencias en lo logrado por los niños; además que ésta ha sido fundamental para desarrollar las investigaciones que estamos realizando actualmente al respecto.

Con el título de "Exploración de Tres Métodos de la Enseñanza del lenguaje LOGO" se encuentra un documento de 35 páginas en la Facultad de Matemáticas de la Pontificia Universidad Católica de Chile.

COMPUTADOR E SOCIEDADE - UMA EXPERIÊNCIA DIDATICA

Cláudia Sabani
Universidade Federal do Rio Grande do Sul
Porto Alegre - *Brasil*

A finalidade deste trabalho é relatar uma experiência de ensino ocorrida no 1º Semestre de 1986: o desenvolvimento da disciplina "Computador e Sociedade", do Curso de Bacharelado em Ciência da Computação da Universidade Federal do Rio Grande do Sul (Brasil). Neste curso, de 8 semestres letivos, a referida disciplina pertence ao 7º Semestre, e é a única que se propõe a discutir os efeitos do uso da informática na sociedade e sobre o indivíduo. Assim sendo, meu objetivo como professora da mesma foi sensibilizar os alunos para uma reflexão mais profunda sobre o impacto da informatização e promover um questionamento sobre as melhores opções para a realidade brasileira.

No decorrer da disciplina procurou-se ter em mente as diferenças do processo de informatização nos países desenvolvidos e nos países em desenvolvimento, especificamente o Brasil. Assim sendo, questionou-se as soluções adotadas pelos países desenvolvidos e sua adequação/inadequação à realidade brasileira.

As atividades do curso poderiam ser divididas em três momentos distintos:

a) Num primeiro momento, o objetivo foi essencialmente o de sensibilizar os alunos para os problemas surgidos com a informatização. Minha preocupação era a de que sendo os alunos (em sua maioria) profissionais da área de computação e que pela primeira vez tinham a oportunidade de discutir em aula este tema (apesar de já estarem em final de curso), estivessem pois voltados apenas para os benefícios da informática, tendo para si apenas uma visão do seu lado positivo. Por isto as primeiras aulas foram uma espécie de "provocação", para despertar nos alunos ou uma atitude de defesa do que julgavam inatacável (e que seria questionado), ou um ponto de partida para discussões abertas sobre os temas propostos, entre eles: uso da informática na educação; a informatização fren

te à realidade nacional (pobreza); a política nacional de informática; problemas decorrentes da automação, etc... Como os assuntos tratados nesta área são sempre polêmicos, procurei oferecer aos alunos material sobre os dois lados das questões (prós e contras). As discussões eram feitas primeiramente em pequenos grupos os quais elaboravam suas conclusões e as lançavam num debate de grande grupo.

b) Com a fase anterior, de discussão em pequenos e grande grupos pretendeu-se desenvolver um espírito crítico e de questionamento. Assim, poderia-se passar para uma segunda fase da disciplina, para a qual foram planejadas palestras com especialistas da área e estudos mais aprofundados sobre os temas propostos.

c) A terceira fase do curso constituiu do desenvolvimento de trabalhos em grupo, orientados no decorrer do semestre e apresentados ao final do mesmo. Os temas escolhidos foram: aplicações da Informática nas diversas áreas (educação, ciências, administração, etc.); impactos da informatização sobre o emprego; Política Nacional de Informática e Indústria Nacional de Informática; Automação Bancária; Automação Comercial; Automação Industrial e Automação de Escritórios.

Durante todo o curso pretendeu-se examinar com clareza e sem mistificações toda a problemática dos efeitos da informatização da sociedade. Procurou-se analisar todas as vantagens trazidas tais como: maior produtividade, melhoria nas condições de trabalho, economia de tempo e liberação de tarefas tediosas, melhoria na qualidade dos produtos obtidos, estímulo à criatividade e a racionalização das tarefas, etc. Porém não se pode esquecer dos aspectos negativos: eliminação de empregos, pela substituição homem/máquina; invasão da privacidade; diminuição do nível salarial da mão-de-obra excedente; produtos e profissões obsoletos a curto prazo, etc.

Concluindo, considero que os objetivos da disciplina foram atingidos pois houve uma conscientização sobre a importância de tais questões, houve um questionamento aberto e franco sobre os problemas nacionais na área da informática e uma busca de soluções que atendam às nossas reais necessidades.

INDICE

GRUPO I

GRUPO II

GRUPO III

GRUPO IV

GRUPO V

COMUNICACIONES